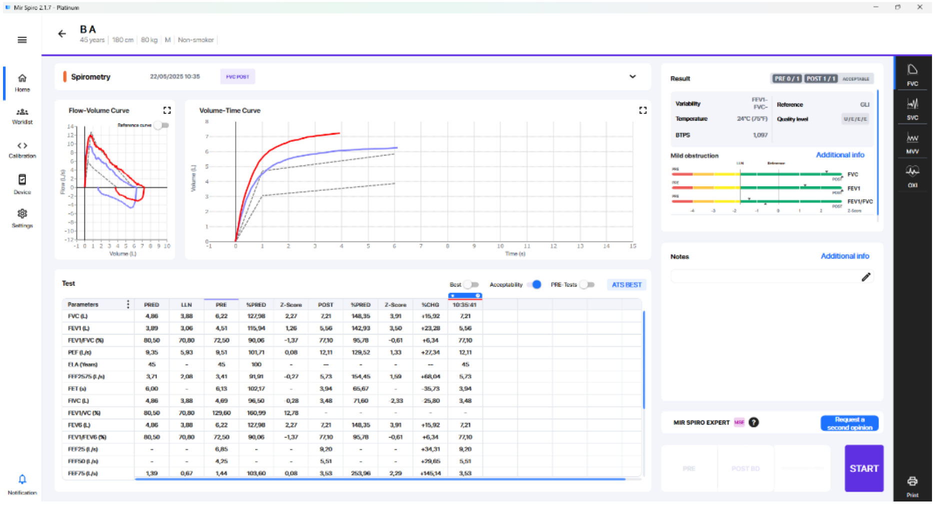
# MIR SUPER SDK
## Spirometry and Oximetry Software Development Kit (SDK)



B A
45 years | 180 cm | 80 kg | M | Non-smoker

Spirometry 22/05/2025 10:35 FVC POST

Result

| | | | |
|---|---|---|---|
| Variability | FEV1-<br>FVC- | Reference | GLI |
| Temperature | 24°C (75°F) | Quality level | U/E/E/E |
| BTPS | 1,097 | | |

Mild obstruction

Notes

Test

| Parameters | PRED | LLN | PRE | %PRED | Z-Score | POST | %PRED | Z-Score | %CHG | 10:35:41 |
|---|---|---|---|---|---|---|---|---|---|---|
| FVC (L) | 4,86 | 3,88 | 6,22 | 127,98 | 2,27 | 7,21 | 148,35 | 3,91 | +15,92 | 7,21 |
| FEV1 (L) | 3,89 | 3,06 | 4,51 | 115,94 | 1,26 | 5,56 | 142,93 | 3,50 | +23,28 | 5,56 |
| FEV1/FVC (%) | 80,50 | 70,80 | 72,50 | 90,06 | -1,37 | 77,10 | 95,78 | -0,61 | +6,34 | 77,10 |
| PEF (L/s) | 9,35 | 5,93 | 9,51 | 101,71 | 0,08 | 12,11 | 129,52 | 1,33 | +27,34 | 12,11 |
| ELA (Years) | 45 | - | 45 | 100 | - | — | - | - | — | 45 |
| FEF2575 (L/s) | 3,71 | 2,08 | 3,41 | 91,91 | -0,27 | 5,73 | 154,45 | 1,59 | +68,04 | 5,73 |
| FET (s) | 6,00 | - | 6,13 | 102,17 | - | 3,94 | 65,67 | - | -35,73 | 3,94 |
| FIVC (L) | 4,86 | 3,88 | 4,69 | 96,50 | -0,28 | 3,48 | 71,60 | -2,33 | -25,80 | 3,48 |
| FEV1/VC (%) | 80,50 | 70,80 | 129,60 | 160,99 | 12,78 | - | - | - | - | - |
| FEV6 (L) | 4,86 | 3,88 | 6,22 | 127,98 | 2,27 | 7,21 | 148,35 | 3,91 | +15,92 | 7,21 |
| FEV1/FEV6 (%) | 80,50 | 70,80 | 72,50 | 90,06 | -1,37 | 77,10 | 95,78 | -0,61 | +6,34 | 77,10 |
| FEF25 (L/s) | - | - | 6,85 | - | - | 9,20 | - | - | +34,31 | 9,20 |
| FEF50 (L/s) | - | - | 4,25 | - | - | 5,51 | - | - | +29,65 | 5,51 |
| FEF75 (L/s) | 1,39 | 0,67 | 1,44 | 103,60 | 0,08 | 3,53 | 253,96 | 2,29 | +145,14 | 3,53 |

| | | |
|---|---|---|
| User manual rev. 0.5 | Issue date | 30.05.2025 |
| | Approval date | 30.05.2025 |

ENGLISH (EN)

CE
2460

# INDEX

# THANK YOU FOR CHOOSING A MIR PRODUCT
MEDICAL INTERNATIONAL RESEARCH

**Before using your device**

- Read carefully your User Manual and pay attention to all the warnings and labels including all relevant information included with the product.

Manufacturer's address:

**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

**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.**

**FEDERAL LAW RESTRICTS THIS DEVICE TO SALE BY OR ON THE ORDER OF A PHYSICIAN**

## 1. Introduction

The MIR SUPER SDK (SSDK) - 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 software developers in the medical field.

| Commercial name | Ref | BASIC-UDI | UDI-DI |
|---|---|---|---|
| MIR SUPER SDK (for Windows) | 920230 | 805299032SSDKHV | 8052990322091 |
| MIR SUPER SDK (for Mac) | 920232 | 805299032SSDKHV | 8052990322220 |
| MIR SUPER SDK (for iOS) | 920234 | 805299032SSDKHV | 8052990322237 |
| MIR SUPER SDK (for Android) | 920236 | 805299032SSDKHV | 8052990322244 |
| MIR SUPER SDK (for Web) | 920238 | 805299032SSDKHV | 8052990322251 |

### 1.1 Intended use

MIR SUPER SDK is a medical device software. It is intended to receive raw spirometry and oximetry data from compatible medical devices. These raw data are processed to provide information on patients' lung function.

### 1.2 Intended users

Qualified healthcare professionals and software developers.

### 1.3 Ability and experience required

When used for a medical purpose, the use of the MIR Super SDK is restricted to qualified healthcare professionals and requires connection to a compatible device along with one of the user interface accessories specified in Section 1.2.
Access to and use of the MIR SUPER SDK for development purposes is restricted to software developers who are properly qualified and trained.

⚠ WARNING

**The manufacturer cannot be held responsible for any damage caused by the user of the device failing to follow instructions and warnings in this manual.**
**If the user of the device is a person considered to be cognitively impaired the operation of the device must be made under the supervision and responsibility of the person legally responsible to supervise the cognitively impaired person.**

⚠ WARNING

**When used as a pulse-oximeter, the device is intended for spot-checking, overnight sleep screening and/or continuous monitoring when used by a trained healthcare professional.**

### 1.4 Intended Population

Spirometry and oximetry are tests that perform measurements of several parameters linked to respiratory function. Whether such measured values correspond to a physiological or pathological condition, it is up to the qualified healthcare professional, who performs the diagnosis, to decide. The Super SDK medical device only processes raw spirometry and oximetry data coming from such measurements.
For these reasons, the target population of the medical device itself is the entire population, except for children under three years as regards spirometry tests. Children under three years are prone to be not compliant to the procedure and to give unreliable results for spirometry tests (as per ATS/ERS 2019 guidelines).

### 1.5 Indications for use

*Spirometry*
Spirometry test is performed on patients to collect sufficient data on the patient lung functions capabilities. The spirometer performs a measurement and, as such, it is not a medical device providing a diagnosis. The numerical information retrieved from spirometry should be integrated by the physician with other clinical and instrumental data to get a proper diagnosis.
Some indications for spirometry are (as reported in ATS/ERS Standardization of spirometry 2019):

1.Diagnosis
  To evaluate symptoms, signs, or abnormal laboratory test results
  To measure the physiologic effect of disease or disorder
  To screen individuals at risk of having pulmonary disease
  To assess preoperative risk
  To assess prognosis
2. Monitoring
  To assess response to therapeutic intervention
  To monitor disease progression
  To monitor patients for exacerbations of disease and recovery from exacerbations
  To monitor people for adverse effects of exposure to injurious agents
  To watch for adverse reactions to drugs with known pulmonary toxicity
3. Disability/impairment evaluations
  To assess patients as part of a rehabilitation program

To assess risks as part of an insurance evaluation
To assess individuals for legal reasons

4. Other
Research and clinical trials
Epidemiological surveys
Derivation of reference equations
Preemployment and lung health monitoring for at-risk occupations
To assess health status before beginning at-risk physical activities.

*Pulse oximetry*

Pulse Oximetry is a non-invasive test that measures the oxygen saturation level (SpO2) in the blood. This test measures how much oxygen is being carried by the haemoglobin in the blood. However, like spirometry, pulse oximetry alone is insufficient for diagnosing the underlying cause of a condition and should be used alongside other clinical evaluations. Some indications for pulse oximetry, are:

- Screening for respiratory conditions: It can help identify individuals at risk of respiratory disorders, such as COPD or asthma, by providing non-invasive measurement of oxygen saturation.
- Preoperative assessment: Pulse oximetry is commonly used to evaluate patients prior to surgery to assess perioperative risk and ensure proper oxygenation during and after anaesthesia.
- Assessing treatment response: Monitoring oxygen saturation during therapeutic interventions helps evaluate the effectiveness of treatments, such as oxygen therapy, in acute and chronic respiratory diseases.
- Tracking disease progression: Continuous or regular monitoring of patients with conditions like COPD, asthma, and pneumonia allows healthcare providers to adjust care based on changes in $SpO_2$ levels.
- Post-operative and ICU care: In critically ill patients or those recovering from surgery, pulse oximetry ensures early detection of hypoxemia, guiding timely interventions.
- Pulse oximetry is often part of assessments in rehabilitation programs, disability claims, and insurance evaluations to determine the impact of respiratory conditions on daily function and work capacity.
- Clinical trials and research: Pulse oximetry is frequently used in research to evaluate respiratory interventions and derive reference values for populations at risk of hypoxemia.
- Workplace health monitoring: It is used in high-risk occupations (e.g., mining, firefighting) to ensure workers are not experiencing dangerous levels of oxygen desaturation.
- Fitness assessments: Pulse oximetry is employed to evaluate oxygen saturation before high-risk physical activities such as mountain climbing or deep-sea diving.

## 1.6 Clinical benefits of the device

***Spirometry:***

As clearly stated in the ATS/ERS guidelines 2019, spirometry is fundamental in the assessment of general respiratory health. Spirometry enables measuring the effect of a disease on lung function, assessing airway responsiveness, monitoring disease course or the result of therapeutic interventions, assessing preoperative risk, and determining a prognosis for many pulmonary conditions. Spirometry is a valuable tool that provides important information to clinicians which is used together with other physical findings, symptoms, and history to reach a diagnosis.

The primary signal measured in spirometry is either volume or flow as a function of time. The most relevant measurements discussed in this document are the FVC, which is the volume delivered during an expiration made as forcefully and completely as possible starting from full inspiration, and the FEV1, which is the expiratory volume in the first second of an FVC maneuver. Other well-established parameters in spirometry include $FEV_{25-75}$, which represents the average expiratory flow between 25% and 75% of the forced vital capacity, and PEF (Peak Expiratory Flow), which indicates the maximum flow achieved during a forced expiration. Although the above mentioned parameters are the primary variables measured in spirometry, there is far more information contained in the flow and volume data. For this reason, a number of parameters have been derived and are taken into account in the ATS/ERS spirometry guidelines. A comprehensive list of the parameters cited and commented in the guidelines are given below:

- VC, FVC, FEV1, PEF, MVV, FEV2575, FEVt (derived from instantaneous volume measurement),
- FEV1/FVC, FEV1/VC, FEF% (derived by instantaneous flow measurement),
- FIF%, FIF2575, FIVC, VC, IVC, EVC, TLC, ERV, Ti, Ttot, FEV1/FEV6, FEF50/FIF50, FEV3/FVC, FEV3/FEV6 (derived from the inspiratory phase and by combining parameters)

Being mentioned and recommended by the guidelines all these parameters are estimated by the subject devices.

Spirometry is a consolidated technique in the diagnosis and management of respiratory diseases, for which it has and continues to have important and numerous clinical benefits.

The clinical benefits of spirometry—as a valuable tool for managing pulmonary and respiratory diseases—can be summarized as follows:

- Diagnosis of lung and respiratory diseases: Helps detect conditions such as asthma, COPD, or restrictive lung diseases at an early stage, often before symptoms become evident.
- Trend assessment (monitoring) of lung function: Allows clinicians to assess the progression of disease or the response to treatment through regular follow-up measurements.
- Symptoms relief: Supports identification of reversible airway obstruction, enabling targeted therapy that can improve symptoms like shortness of breath or wheezing.
- Assessment of disease severity: Provides objective data to classify the severity of a respiratory condition, which is essential for treatment planning and prognosis.
- Driving the therapeutic management: Informs clinical choices regarding medication adjustments, initiation of inhaled therapies, or need for further investigation.
- Reduction in exacerbations: Enables timely interventions that can lower the frequency and intensity of disease flare-ups.

- Reduction in hospital admissions: Better disease control through spirometry-guided management helps prevent complications requiring hospitalization.
- Prevention of disease progression: Early and accurate monitoring can prompt interventions that slow down or halt the worsening of lung function.
- Improvement in quality of life: Enhanced symptom control and stability lead to better daily functioning and overall well-being.
- Increased of survival: Optimized disease management and prevention of severe events contribute to longer life expectancy in patients with chronic respiratory conditions.

***Oximetry:***

Pulse oximetry has become a common practice in a variety of clinical situations including the assessment and management of all patients at risk of respiratory dysfunction. Adequate oxygen content in arterial blood guarantees satisfactory tissue perfusion and provides critical information about lung function. Pulse oximetry is a non-invasive method that enables rapid measurement of the oxygen saturation of hemoglobin in arterial blood. It can rapidly detect changes in oxygen
saturation, thus providing an early warning of dangerous hypoxemia.

Oximetry is thus a consolidated technique in the management of acute and chronic respiratory disease for which it has and continues to have important and numerous clinical benefits.

The clinical benefits of oximetry—as a valuable tool for managing pulmonary and respiratory diseases—can be summarized as follows:

- Part of clinical assessment for suspected respiratory diseases: Provides immediate information about blood oxygen saturation, helping clinicians identify potential respiratory conditions and guiding further diagnostic steps.
- Assessment of disease severity: Provides objective data to classify the severity of a respiratory condition, which is essential for treatment planning and prognosis.
- Optimized use of oxygen therapy: Helps tailor oxygen flow rates based on real-time $SpO_2$ levels, avoiding unnecessary use or insufficient oxygen delivery.
- Follow-up of patients after a severe or complicated exacerbation: help in tracking oxygen levels over time, assess recovery progress, and detect any signs of relapse or persistent hypoxemia, allowing timely medical intervention if needed.
- Prevention of disease progression: Early and accurate monitoring can prompt interventions that slow down or halt the worsening of lung function.
- Reduced risk of complications and hospital admissions: Continuous or periodic monitoring allows for timely intervention, helping to prevent severe episodes that might otherwise require emergency care or hospitalization.
- Support during physical activity: allows patients to safely engage in daily activities or rehabilitation by monitoring oxygen levels during exertion and adjusting pace or oxygen use accordingly.

## 1.7 Essential performances

The MIR SUPER SDK has two main essential performances:

- Spirometry data treatment
- Oximetry data treatment

The accuracy related to the processing of these data corresponds to that of the devices compatible with the SSDK.

## 1.8 Operating Environment

When used by healthcare professional for a medical purpose, Super SDK has been designed for use in hospital setting, physician's office, factory, pharmacy.

## 1.9 Limitations of use - Contraindications

### Oximetry limitations

*Measurement Accuracy:*

Motion Artifacts: Movement by the patient can cause inaccurate readings. This is especially problematic in children or restless patients.

Peripheral Vasoconstriction: Conditions that reduce blood flow to the extremities, such as cold temperatures or shock, can affect the accuracy of the readings.

Nail Polish and Artificial Nails: These can interfere with the light transmission used in pulse oximetry, leading to incorrect readings.

Skin Pigmentation: Variations in skin color can potentially affect the accuracy of oximetry readings. Specifically overestimation in individuals with darker skin is well known and addressed at the regulatory level by specific tests. For the oximetry sensors that can be used with SSDK the $SpO_2$ readings resulted to be not affected by skin pigmentation.

*Technical Limitations:*

Calibration Requirements: Oximeters require regular calibration to maintain accuracy, which can be a limitation in resource-limited settings.

Battery Dependency: Portable oximeters are reliant on battery power, and power depletion can lead to device failure during critical monitoring.

Signal Interference: External light sources and electromagnetic interference can affect the performance of pulse oximeters.

*Clinical Limitations:*

Limited Diagnostic Capability: While oximetry can indicate hypoxemia, it does not provide information on the underlying cause. Further diagnostic tests are usually required.

False Reassurance: Patients with normal oximetry readings might still have significant respiratory or cardiovascular conditions that are not detected by oximetry alone.

Sensor Placement: Proper sensor placement is crucial, and incorrect placement can lead to false readings.

*Physiological Limitations:*

Carbon Monoxide Poisoning: Pulse oximeters cannot differentiate between oxygen and carbon monoxide-bound hemoglobin, which can result in falsely elevated readings.
Methemoglobinemia: This condition can also lead to inaccurate oximetry readings, as methemoglobin absorbs light differently than normal hemoglobin.

**Spirometry limitations**
*Measurement Accuracy:*
Effort Dependency: Accurate spirometry results depend on the patient's effort and ability to follow instructions. Poor effort can lead to suboptimal results.
Variability: Spirometry measurements can vary significantly between tests and require consistent technique to ensure reliability.
*Technical Limitations:*
Calibration and Maintenance: Spirometers require regular calibration and maintenance to ensure accurate measurements. This can be resource-intensive.
Environmental Factors: Temperature, humidity, and altitude can affect the accuracy of spirometry measurements, necessitating adjustments or corrections.
*Clinical Limitations:*
Limited Use in Certain Populations: Spirometry is challenging in very young children (under 3 years) and individuals with severe cognitive or physical impairments who cannot perform the required maneuvers.
Inability to Measure All Aspects of Lung Function: Spirometry primarily measures airflow and lung volumes but does not provide information on gas exchange or lung tissue pathology.
Patient Condition: Acute illnesses, recent surgeries, or severe respiratory distress can prevent patients from performing spirometry, limiting its applicability in these contexts.
*Interpretation Challenges:*
Complex Interpretation: Accurate interpretation of spirometry results requires expertise and experience. Variability in interpretation can occur among clinicians.
Predicted Values: Spirometry relies on predicted values based on population averages (age, sex, height, and ethnicity). Predicted values are routinely used, healthcare professionals are adequately trained to interpret the results appropriately in the context.

**Contraindications**
MIR SSDK does not have direct contraindications for use. Typically, contraindications for the use of spirometers would align with contraindications for the underlying medical procedures or interventions being performed.

*Spirometry*
Spirobank II PLUS and Spirobank II Light spirometers have the relative contraindications of the spirometry, as reported in the 2019 update of the ATS/ERS guideline. According to this guideline, performing spirometry can be physically demanding. The forced expiratory maneuver used in spirometry increases intrathoracic, intraabdominal, and intracranial pressures. Potential risks of spirometry are primarily related to maximal pressures generated in the thorax and their impact on abdominal and thoracic organs, venous return and systemic blood pressure, and expansion of the chest wall and lung. The physical effort required can increase myocardial demand.
For these reasons the ATS/ERS guidelines recommends to use caution for patients with medical conditions that could be adversely affected by these physiological consequence. The medical conditions for which caution is recommended are reported below:

1. Due to increases in myocardial demand or changes in blood pressure
   - Acute myocardial infarction within 1 wk
   - Systemic hypotension or severe hypertension
   - Significant atrial/ventricular arrhythmia
   - Noncompensated heart failure
   - Uncontrolled pulmonary hypertension
   - Acute cor pulmonale
   - Clinically unstable pulmonary embolism
   - History of syncope related to forced expiration/cough
2. Due to increases in intracranial/intraocular pressure
   - Cerebral aneurysm
   - Brain surgery within 4 wk
   - Recent concussion with continuing symptoms
   - Eye surgery within 1 wk
3. Due to increases in sinus and middle ear pressures
   - Sinus surgery or middle ear surgery or infection within 1 wk
4. Due to increases in intrathoracic and intraabdominal pressure
   - Presence of pneumothorax
   - Thoracic surgery within 4 wk
   - Abdominal surgery within 4 wk
   - Late-term pregnancy
5. Infection control issues
   - Active or suspected transmissible respiratory or systemic infection, including tuberculosis
   - Physical conditions predisposing to transmission of infections, such as hemoptysis, significant secretions, or oral lesions or oral bleeding.

Although such risks are likely to be minimal for spirometry in most patients, the potential risks associated with testing should always be weighed against the benefit of obtaining information about lung function. Spirometry should be discontinued if the patient experiences pain during the maneuver.

Patients with potential contraindications that would prevent testing in the primary care setting may be tested in a pulmonary function laboratory where operators are more experienced and there may be access to emergency care if needed.

Furthermore, since spirometry requires the active participation of the patient, inability to understand directions, unwillingness to follow the directions of the operator will usually lead to submaximal test results (ATS/ERS guideline).

A significant quantitative data is reported in a 20-year review of 186,000 pulmonary function tests in a tertiary institution (Roberts C et al, Thorax 2018;73:385–387). It was found that patient safety incidents occurred in 5 of every 10,000 routine pulmonary function tests (excluding exercise and provocation tests) with generally low risk of harm. The most common finding was primarily syncope.

*Pulse oximetry*

There are no contraindications to pulse oximetry. It is generally safe to use in monitoring all patients.

Although the pulse oximeter is generally a safe device, its use still carries some risk of adverse events. Burning or blistering at the placement site may occur if the light emitting diode becomes overheated. Ischemic pressure necrosis may result if the probe is placed too tightly on the patient. Perioperative corneal abrasion may occur if a patient with a probe on a finger rubs his or her eyes when awakening from anesthesia and in doing so scrapes the cornea. Prolonged placement of a pulse oximeter probe, which can occur in patients in the ICU, may lead to mechanical injury, such as finger stiffness, making it difficult for the patient to flex the finger after the probe has been removed. Although these complications have been documented, they remain uncommon.

Pulse oximetry is a non-invasive technology with limited potential to produce adverse events. However, digital injury occasionally can occur when continuous pulse oximetry is applied for several days; this complication appears more common in patients receiving vasopressor therapy. Other limitations include the inability to measure ventilation or the partial pressure of arterial carbon dioxide, and the potential for delay in detection of acute hypoxemia. In neonates, pulse oximetry is also unable to detect significant hyperoxia, which can result in oxygen toxicity. Common sources of artifact include inappropriate probe placement, motion artifact, ambient light, electromagnetic radiation, low blood flow to the sensor area.

Monitoring of heart rate and SpO2 is recommended in the Pulmonary Rehabilitation Toolkit (www.pulmonaryrehab.com.au) for safety (including determination of contraindications to exercise and the need for supplemental oxygen) and to determine the patient’s physiological response to exercise, which may assist prescription and progression.

### 1.10 Compatible devices and accessories

The list of the SSDK compatible devices is:

- Minispir and Minispir Light (Spirometer)
- Spirobank II Basic and Spirobank II with Bluetooth Smart (Spirometer and optional oximeter)
- Spirolab (Spirometer and optional oximeter)
- Spirodoc (Spirometer and optional oximeter)
- Spirobank Smart and Spirobank Oxi (Spirometer and oximeter, which are part of the Smart family products).
- Minispir PLUS and Minispir PLUS ESI (Spirometer)
- Spirobank II PLUS and Spirobank II Light (Spirometer and optional oximeter)
- Spirolab PLUS and Spirolab PLUS ESI (Spirometer and optional oximeter)

They are MD class IIa, compatible with MIR SUPER SDK. MIR is the legal manufacturer of all SSDK compatible devices (see DoC, IFU, and General Description for details). They allow the user to perform spirometry and oximetry tests.

None of the products are supplied to the end user along with the SSDK.

The SSDK is only compatible with these devices because communication between the device and the platform (PC, Tablet, Smartphone) is done using proprietary messages (called "Protocols") developed by MIR. The SSDK is specifically designed to recognize the proprietary protocol of MIR devices and, therefore, can only communicate with them.

Similarly, without the SSDK, communication with the device is not possible because, without the proprietary protocol, the exchange of data cannot occur.

The list of accessories of the SSDK (user interfaces that provide features such as data visualization, patient management, and reporting) is:

- MIR Spiro (Base and Platinum)
- MIR Spiro Expert
- Pneumotel Web and Pneumotel App.

None of the products are supplied to the end user along with the SSDK.

### 1.11 Important safety warnings

The MIR SSDK is continuously checked during manufacturing and therefore the product complies with the established security levels and quality standards laid down by Regulation (EU) 2017/745 for medical devices.

⚠ WARNING

**The safety and the correct performance of the device can only be assured if the user respects all of the relevant safety rules and regulations. The manufacturer will not be held responsible for damage due to user's neglect to correctly to follow these instructions.**
**The device must be used following the indications given by the manufacturer with particular attention to the paragraph on INTENDED USE, and utilizing only the compatible devices and the accessories listed in par. 1.2. Use of not-compatible devices and accessories may cause errors in measurement and/or compromise the correct functioning of the device, and is therefore not permitted.**
**The device should not be used beyond the declared life time. In normal conditions the life time of the device has been established in advance to be 10 years. Please see par. 1.12 for further details.**

**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.**

- A spirometry test should only be carried out when the patient is at rest and in good health, in suitable testing conditions. A spirometry test requires the full collaboration of the patient since she/he must perform a complete forced expiration, in order to obtain a reliable test result.
- The MIR SUPER SDK is exclusively intended for the measurement of lung function parameters and peripheral oxygen saturation. Any other uses are not recommended.
- 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 prototcol.
- 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 (privacy): Please note that MIR SUPER SDK does not use any personal user information.

The management of security for the SSDK is the responsibility of the user. Users are required to actively secure the software by employing comprehensive antivirus solutions and consistently updating security protocols. This responsibility extends to any associated hardware or software that interacts with the SSDK. It is crucial that all components are adequately protected from potential threats to maintain the system's integrity and functionality.

### 1.12 Lifetime

The software lifetime of the MIR SUPER SDK may vary. However, a minimum usage period of 10 years has been established in advance, starting from the date indicated on the label. This is because medical device (MD) software code does not physically degrade over time. However, there are several limitations that may affect this period:

- Manufacturer's decision: The manufacturer may choose to discontinue support or use of the software before the end of the expected lifetime.
- Loss of essential performance due to changes in the operating environment: For example, updates or changes in operating systems such as MacOS, iOS, Windows, or Android may cause the software to malfunction or become incompatible.
- End of support for the programming languages used in the SDK: If the development languages used for the MIR SUPER SDK are no longer supported, further development or maintenance may not be possible.
- Regulatory authority decisions: A regulatory body may decide to stop the distribution of the medical device software for safety or compliance reasons.

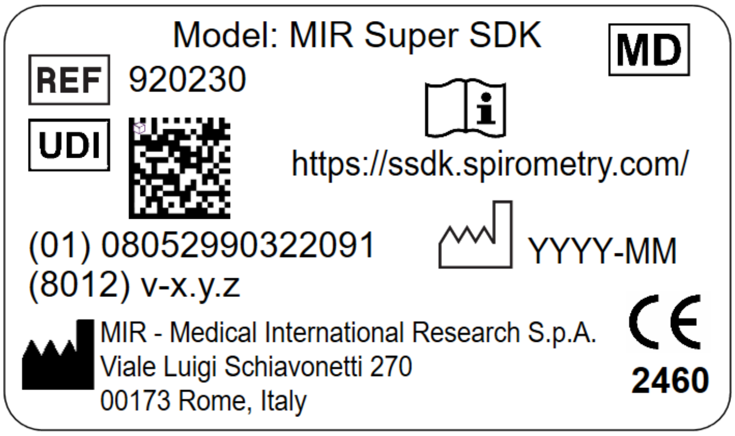
In the event that MIR decides to discontinue the distribution and use of its Medical Device Software (MDSW), the following steps are taken:

- An Advisory Notice is sent to all distributors.
- Users are informed through the associated MDSW that the software will be withdrawn from the market after a pre-defined period.

Note. The MIR SUPER SDK does not store any sensitive data.

### 1.13 Labels and symbols

### 1.13.1 Identification label and symbols



Model: MIR Super SDK

MD

REF 920230

UDI

https://ssdk.spirometry.com/

(01) 08052990322091
(8012) v-x.y.z

YYYY-MM

MIR - Medical International Research S.p.A.
Viale Luigi Schiavonetti 270
00173 Rome, Italy

CE
2460

The symbols present in the device labels are described in the table below:

| 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 |
| CE<br>2460 | This product is certified CE to conform to the Class IIa requirements of the European Regulation (EU) 2017/745. |
| ℹ | eIFU indicator |
| 🏭 | Manufacturing date: year and month of software version release date. |

## 2. Product description

The MIR SUPER SDK is a software designed for medical use in the field of spirometry. It serves as a comprehensive solution that processes and analyzes spirometry and oximetry data, providing accurate respiratory measurements for medical evaluation. With real-time feedback and data visualization features, healthcare professionals can efficiently interpret and assess the obtained results.

The MIR Super SDK is specifically designed for clinical use. It can also function as a 'black box' module, allowing integration with other software applications without requiring access to its internal mechanisms. It provides essential functionality related to the assessment of the patient lung function, including data acquisition, processing, analysis, interpretation and transmission of spirometry and oximetry parameters.

The SSDK is developed by MIR, a recognized medical device software provider. It is designed and manufactured in compliance with applicable regulatory requirements and standards, ensuring its safety, effectiveness, and performance as a medical device. The SSDK undergoes a comprehensive development process, including design, implementation, verification, and validation, to meet the necessary quality standards.

The MIR SSDK Accessory software, developed by MIR, includes MIR SPIRO (Base and Platinum), MIR SPIRO EXPERT, PNEUMOTEL WEB, and PNEUMOTEL APP (par. 1.10). These software applications utilize the SSDK as the core component. They serve as user interfaces, providing features such as data visualization, patient management, and reporting.

The MIR Software Platform, including the SSDK and MIR SSDK Accessory software, aims to enhance respiratory care by providing reliable and accurate measurement, analysis and transmission of spirometry and oximetry parameters. It enables qualified healthcare professionals to monitor and manage patients' respiratory health remotely, facilitating early detection of respiratory conditions and supporting informed clinical decision-making.

### 2.1 Technical specification

For the spirometry, MIR SSDK displays the spirometry parameters listed below as they are sent by the compatible spirometer.

| Symbol | Description | Units |
|---|---|---|
| *FVC | Best FVC | L |
| *FEV1 | Best FEV1 | L |
| *PEF | Best PEF | L/s |
| FVC | Forced Vital Capacity | L |
| FEV1 | Volume expired in the 1st second of the test | L |

| | | |
|---|---|---|
| FEV1/FVC | FEV1/FVC x 100 | % |
| FEV1/VC | FEV1 / best between EVC and IVC x 100 | % |
| PEF | Peak expiratory flow | L/s |
| FEF2575 | Average flow between 25% and 75% of the FVC | L/s |
| FEF25 | Forced Expiratory Flow at 25% of FVC | L/s |
| FEF50 | Forced Expiratory Flow at 50% of FVC | L/s |
| FEF75 | Forced Expiratory Flow at 75% of FVC | L/s |
| FEV3 | Volume expired in the initial 3 seconds of the test | L |
| FEV3/FVC | FEV3/FVC x 100 | % |
| FEV6 | Volume expired in the initial 6 seconds of the test | L |
| FEV6% | FEV1/FEV6 x 100 | % |
| FEV075 | Volume expired after 0.75 seconds after extrapolate volume | L |
| FET | Forced expiratory time | s |
| BEV | Back Extrapolated Volume | L |
| FIVC | Forced inspiratory volume | L |
| FIV1 | Volume inspired in the 1st second of the test | L |
| FIV1/FIVC | FIV 1 % | % |
| PIF | Peak inspiratory flow | L/s |
| MVVcal | Maximum voluntary ventilation calculated on FEV1 | L/s |
| VC | Slow vital capacity (the best value between expiratory and inspiratory) | L |
| EVC | Slow expiratory vital capacity | L |
| IVC | Slow inspiratory vital capacity | L |
| IC | Inspiratory capacity (max between EVC and IVC) - ERV | L |
| ERV | Expiratory reserve volume | L |
| TV | Current volume | L |
| RR | Respiratory frequency | Breath/min |
| $t_I$ | Average time of inspiration, at rest | s |
| $t_E$ | Average time of expiration, at rest | s |
| $TV/t_I$ | Average flow of inspiration, at rest | L/min |
| $t_I/t_{TOT}$ | $t_I/(t_I+t_E)$ | \ |
| MVV | Maximum voluntary ventilation | L/min |
| ELA | Estimated lung age | year |
| PEFTime | Time to perform the 90% of PEF | S |
| FEV05 | Volume expired in the initial 0.5 seconds of the test | L |
| FEV05/FVC | FEV05/FVC x 100 | % |
| FEV075/FVC | FEV075/FVC x 100 | % |
| FEV2 | Volume expired in the initial 2 seconds of the test | L |
| FEV2/FVC | FEV2/FVC x 100 | % |
| FEF7585 | Average flow between 75% and 85% of the FVC | L/s |
| FIF25 | Maximum flow at 25% of FIVC | L/s |
| FIF50 | Maximum flow at 50% of FIVC | L/s |
| FIF75 | Maximum flow at 75% of FIVC | L/s |
| R50 | FEF50/FIF50 x 100 | % |
| EI | FEV1/PEF | / |
| RFEV | FEV1/FEV05 | / |
| IRV | Inspiratory reserve volume | L |
| te/ti | Average time of expiration, at rest / Average time of inspiration, at rest | / |
| MV | Minute Volume | L |

- *= best values

In addition to displaying spirometry parameters as estimated by the connected device, the SSDK also implements other formulas, starting from the spirometry parameter values listed above and the patient's anthropometric data, to provide other indications regarding the predicted values and the acceptability and reproducibility of the tests, as reported below:

| Information provided by the SSDK | Meaning |
|---|---|
| Spirometry predicted values | Predicted values of the measured spirometry parameters |
| Spirometry LLN values | Low Limit Normal values of the measured spirometry parameters |
| Spirometry % of predicted | For each spirometry parameter, the % amount respect to the Predicted values |
| Spirometry zScore | Z Scores of the measured spirometry parameters |
| Interpretation as per ATS standard | Obstruction and/ or restriction with the following severity grades:<br>- Mild Obstruction<br>- Moderate Obstruction<br>- Severe Obstruction<br>- Very Severe Obstruction<br>- Mild Restriction<br>- Moderate Restriction<br>- Severe Restriction<br>- Very Severe Restriction |
| Acceptability of the measurements as per ATS standard | If one or more errors in performing the spirometry maneuvers occurred, one or more related messages are provided:<br>-abrupt stop<br>-cough in first second of expiration<br>-no plateau<br>-cough in first second of expiration<br>-slow start<br>-hesitant start<br>-hesitation at maximum volume<br>-slow filling<br>-low final inspiration<br>-incomplete inspiration prior to FVC |
| Quality Control Grade | Categorization of the quality of the spirometry test. The score goes from A (good) to F (poor) |
| Best Trial | Best Trial of a Spirometry Session |
| Session best values | BEST FVC, FEV1, PEF, FEV1/FVC of the Spirometry session |
| FVC variability | Difference between the 2 highest FVC acceptable values of the session |
| FEV1 variability | Difference between the 2 highest FEV1 acceptable values of the session |
| Turbine Calibration Check | Pass or Fail when the the spirometer is connected to a 3L syringe |
| Real Time FVC PRE | Flow/Volume real-time graph |
| Graphical representation of the measured parameters respect to the reference values for FVC FEV1 PEF (zscore based, pictograms) | Values represented on a graphs (pictograms) respect to reference values |

For oximetry, the SSDK takes the SpO2 and heart rate values as input and provides the parameters in the following table as output:

| Symbol | Description | Units |
|---|---|---|
| %SPO2 min | Minimum SPO2 during the test | % |
| %SPO2 max | Maximum SPO2 during the test | % |
| BPM min | Minimum BPM during the test | BPM |
| BPM max | Maximum BPM during the test | BPM |
| %SPO2 mean | Average SPO2 | % |
| BPM mean | Average BPM | BPM |
| T Total | Total test time | hh:mm:ss |
| T Analysis | Total measurement time (test time minus the zeros) | hh:mm:ss |
| T<90% | Time with SPO2 below 90% | %-hh:mm:ss |
| T<89% | Time with SPO2 below 89% | %-hh:mm:ss |
| T<88% | Time with SPO2 below 88% | %-hh:mm:ss |
| T<87% | Time with SPO2 below 87% | %-hh:mm:ss |
| Ev%SPO2<89 | SpO2 fall below 89% for at least 20 seconds | / |
| Δ Index | SpO2 Fluctuation index calculated on 12 second intervals | / |
| T<40BPM | Test time with pulse rate <40 BPM | %-hh:mm:ss |
| T>120BPM | Test time with pulse rate >120 BMP | %-hh:mm:ss |
| Ev<40BPM | Bradycardia (low heart rate) events, during the analysis period | / |
| Ev>120BPM | Tachycardiac (high heart rate) events, during the analysis period | / |
| %SPO2 start | Initial phase %SpO2 base value, before walking test | % |
| %SPO2 end | Final SPO2 reading during walking phase | % |
| BPM start | Initial phase BMP base value, before walking test | BPM |
| BPM end | Final BPM reading during walking phase | BPM |
| T Baseline | Duration of base phase | hh:mm:ss |
| T Walking | Duration of walking phase | hh:mm:ss |
| T Recovery | Duration of recovery phase | hh:mm:ss |
| T2%Δ SPO2 | Time spent during the walking test, with SpO2<2% compared to the SpO2 base value | hh:mm:ss |
| T4%Δ SPO2 | Time spent during the walking test, with SpO2<4% compared to the SpO2 base value | hh:mm:ss |
| Predicted Distance | Standard predicted distance | m |
| Min Predicted Distance (LLN) | Minimum predicted distance | m |
| % Predicted Distance | % variation of the distance covered compared to the standard predicted distance | % |
| % Min Predicted Distance (LLN) | % variation of the distance covered compared to the minimum predicted distance | % |
| AUC/Distance* | Area below the SpO2 base curve compared to the distance covered | / |
| Steps | Estimation of the steps taken by the patient during the test | / |
| VMU** | Number of movements made by the patient during the test | / |
| O2-GAP*** | Estimation of the percentage of oxygen to administer to the patient | % |
| SPO2 Base | SPO2 base value for the SPO2 and ODI tests | % |
| BPM Base | BPM base value for the SPO2 and ODI tests | BPM |
| ODI | Desaturation events per hour of analysis | 1/h |
| Mean Dur. Desat. | Average duration of the desaturation event | s |
| Tot Desaturat./Low SpO2 events | Number of desaturation events during the entire analysis period | / |
| Longest Desat. | Duration of the longest desaturation event | s |

| | | |
|---|---|---|
| Desatur. Peak | Minimum SpO2 value during a desaturation event | % |
| BPM Index | Number of events of variation of the Pulse rate per hour of a analysis | / |
| Mean Desaturat. | Average of the desaturation troughs | s |
| Mean Drop | Average SpO2 fall compared to the base value during the desaturation events | s |
| Max Drop | Maximum fall in the SpO2 compared to the saturation events | s |
| BPM Variation | Number of variations in the Pulse Rate during the entire analysis period | / |
| NOD4% | Number of events with SpO2<4% compared to the SpO2 base value for a continuous period of at least 5 minutes | / |
| NOD89% | Number of events with SpO2<89% for a continuous period of at least 5 minutes | / |
| NOD90% | Number of events with SpO2<90% for a continuous period of at least 5 minutes with min value <86% (Nadir) | / |
| t.NOD4% | Number of events with SpO2<4% compared to the SpO2 base value for a continuous period of at least 5 minutes | hh:mm:ss |
| t.NOD89% | Number of events with SpO2<89% for a continuous period of at least 5 minutes | hh:mm:ss |
| t.NOD90% | Number of events with SpO2<90% for a continuous period of at least 5 minutes with min value <86% (Nadir) | hh:mm:ss |

The following parameters are given as input to finalize the test report:

| | | |
|---|---|---|
| T Total | Total test time | hh:mm:ss |
| T Analysis | Total measurement time (test time minus the zeros) | hh:mm:ss |
| T Baseline | Duration of base phase | hh:mm:ss |
| T Walking | Duration of walking phase | hh:mm:ss |
| T Recovery | Duration of recovery phase | hh:mm:ss |
| Tot Distance Walked | Distance covered | m |
| Dyspnea Base | Degree of breathlessness prior to the walking test | Borg |
| Dyspnea End | Degree of breathlessness at the end of the walking test | Borg |
| Dyspnea CHG | Variation in the degree of breathlessness during the walking test | / |
| Fatigue Base | Degree of tiredness prior to the walking test | Borg |
| Fatigue End | Degree of tiredness at the end of the walking test | Borg |
| Fatigue CHG | Variation in the degree of tiredness during the walking test | / |
| Diastolic Base | Starting diastolic value | mmHg |
| Systolic Base | Starting systolic value | mmHg |
| Diastolic Fine | Final diastolic value | mmHg |
| Systolic Fine | Final systolic value | mmHg |
| O2 | percentage of oxygen administered to the patient before test | L/min-% |

### 2.1.1 Key Functional Elements (KFE)

The MIR SUPER SDK encompasses several key functional elements that enable its effective operation and integration into software applications. These key functional elements of the MIR SUPER SDK work together to enable the seamless integration of the SDK into software applications, allowing the processing, analysis, and utilization of medical data for the intended medical purpose.
Here is a list of the key functional elements:

- Data Acquisition: The MIR SUPER SDK provides capabilities to acquire and collect medical data from compatible devices, sensors, or input sources.
- Data Processing: It includes algorithms and processing capabilities to analyze, filter, and manipulate the acquired data to derive meaningful insights and measurements.
- Medical Calculations: The SDK incorporates medical algorithms and formulas to perform calculations and generate results relevant to the intended medical purpose, such as respiratory parameters, lung function metrics, or diagnostic indicators.
- Data Integration: The MIR SUPER SDK offers mechanisms to seamlessly integrate the processed medical data into external software applications or systems for further analysis, visualization, or reporting.
- Data Security and Privacy: It incorporates measures to ensure the confidentiality, integrity, and privacy of the medical data, adhering to applicable data protection standards and regulations.
- Error Handling and Exception Management: The SDK includes mechanisms to handle errors, exceptions, and unexpected situations, providing appropriate notifications or alerts to users or developers.
- Documentation and Support: The MIR SUPER SDK is accompanied by comprehensive documentation, including API references, integration guidelines, and user manuals, to assist developers in the integration process. It may also provide technical support channels for addressing any queries or issues.

### 2.1.2 Other features

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.

## 2.2 Functioning of the SSDK

When used for medical purposes by a qualified healthcare professional, the software—when connected to compatible devices—performs the following actions via the user interface:

- Start a test
- Store the data related to tests performed on the patient with the devices in "stand alone" mode or make tests in "real time" mode. The collection of tests performed by each patient is organized into separate sessions, enabling result comparison.
- Graphic presentation of parameters relating to human respiratory function. Creation of a clinical history for each patient, allowing test comparisons and efficient data management.
- Print and export reports to be sent to the doctor or healthcare facility for further sharing of information.
- Import of databases from previous archiving, in order not to lose data and maintain the information updated.
- Sharing of data through integration with other IT environment or systems, in order to allow the data to be further processed in various situations.
- Calibration check on the device, according to ATS requirements for the doctor to be able to periodically verify the device operability.
- Various settings configuration, for the doctor to personalize the user experience of the data management.
- Built-in firmware update functionality, in order for released updates to be easily transferred to the user, for bugs and other fixes to be readily available, and to maintain up-to-date functionalities to the on-field devices.
- Web application module for access and visualization of data through user browsers.

For use by software developers, please consult the appendices, which provide detailed instructions for integration within various development environments.

## 2.3 Required resources

For SSDK use with a user interface:

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 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 >11
- 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 12.0 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 6.0
●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)

For software developers:

*Windows Computer (for Windows and Android developments)*
Minimum system requirements:

- OS: Windows 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.

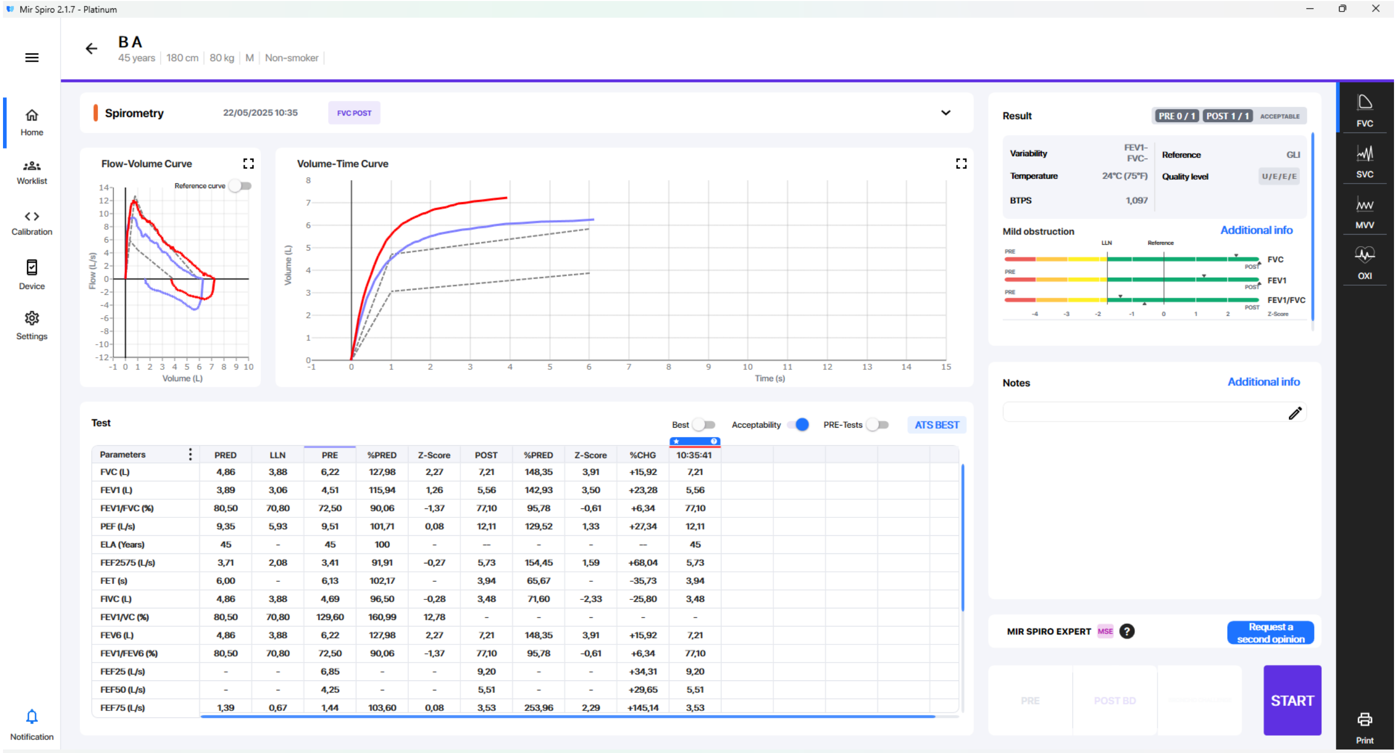
*MacOS computers (For MacOS and iOS developments)*
Minimum system requirements:

- 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.

# 3. Maintenance



B A
45 years | 180 cm | 80 kg | M | Non-smoker

Spirometry 22/05/2025 10:35 FVC POST

Result PRE 0 / 1 POST 1 / 1 ACCEPTABLE

| | | | |
|---|---|---|---|
| Variability | FEV1-<br>FVC- | Reference | GLI |
| Temperature | 24°C (75°F) | Quality level | U/E/E/E |
| BTPS | 1,097 | | |

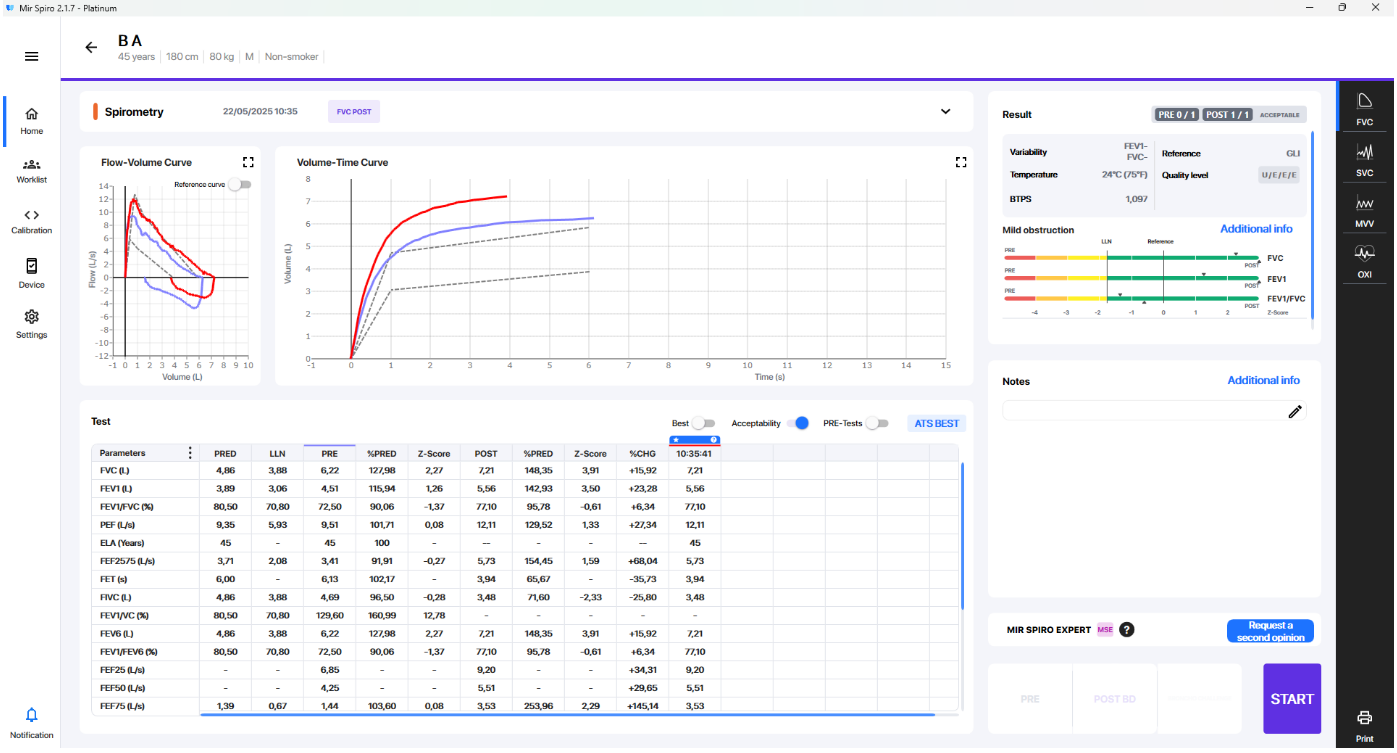
Mild obstruction Additional info

Notes Additional info

MIR SPIRO EXPERT MSE Request a second opinion

Test Best Acceptability PRE-Tests ATS BEST

| Parameters | PRED | LLN | PRE | %PRED | Z-Score | POST | %PRED | Z-Score | %CHG | 10:35:41 |
|---|---|---|---|---|---|---|---|---|---|---|
| FVC (L) | 4,86 | 3,88 | 6,22 | 127,98 | 2,27 | 7,21 | 148,35 | 3,91 | +15,92 | 7,21 |
| FEV1 (L) | 3,89 | 3,06 | 4,51 | 115,94 | 1,26 | 5,56 | 142,93 | 3,50 | +23,28 | 5,56 |
| FEV1/FVC (%) | 80,50 | 70,80 | 72,50 | 90,06 | -1,37 | 77,10 | 95,78 | -0,61 | +6,34 | 77,10 |
| PEF (L/s) | 9,35 | 5,93 | 9,51 | 101,71 | 0,08 | 12,11 | 129,52 | 1,33 | +27,34 | 12,11 |
| ELA (Years) | 45 | - | 45 | 100 | - | -- | - | - | -- | 45 |
| FEF2575 (L/s) | 3,71 | 2,08 | 3,41 | 91,91 | -0,27 | 5,73 | 154,45 | 1,59 | +68,04 | 5,73 |
| FET (s) | 6,00 | - | 6,13 | 102,17 | - | 3,94 | 65,67 | - | -35,73 | 3,94 |
| FIVC (L) | 4,86 | 3,88 | 4,69 | 96,50 | -0,28 | 3,48 | 71,60 | -2,33 | -25,80 | 3,48 |
| FEV1/VC (%) | 80,50 | 70,80 | 129,60 | 160,99 | 12,78 | - | - | - | - | - |
| FEV6 (L) | 4,86 | 3,88 | 6,22 | 127,98 | 2,27 | 7,21 | 148,35 | 3,91 | +15,92 | 7,21 |
| FEV1/FEV6 (%) | 80,50 | 70,80 | 72,50 | 90,06 | -1,37 | 77,10 | 95,78 | -0,61 | +6,34 | 77,10 |
| FEF25 (L/s) | - | - | 6,85 | - | - | 9,20 | - | - | +34,31 | 9,20 |
| FEF50 (L/s) | - | - | 4,25 | - | - | 5,51 | - | - | +29,65 | 5,51 |
| FEF75 (L/s) | 1,39 | 0,67 | 1,44 | 103,60 | 0,08 | 3,53 | 253,96 | 2,29 | +145,14 | 3,53 |


B A
45 years | 180 cm | 80 kg | M | Non-smoker

Spirometry 22/05/2025 10:35 FVC POST

Result PRE 0 / 1 POST 1 / 1 ACCEPTABLE

| | | | |
|---|---|---|---|
| Variability | FEV1-<br>FVC- | Reference | GLI |
| Temperature | 24°C (75°F) | Quality level | U/E/E/E |
| BTPS | 1,097 | | |

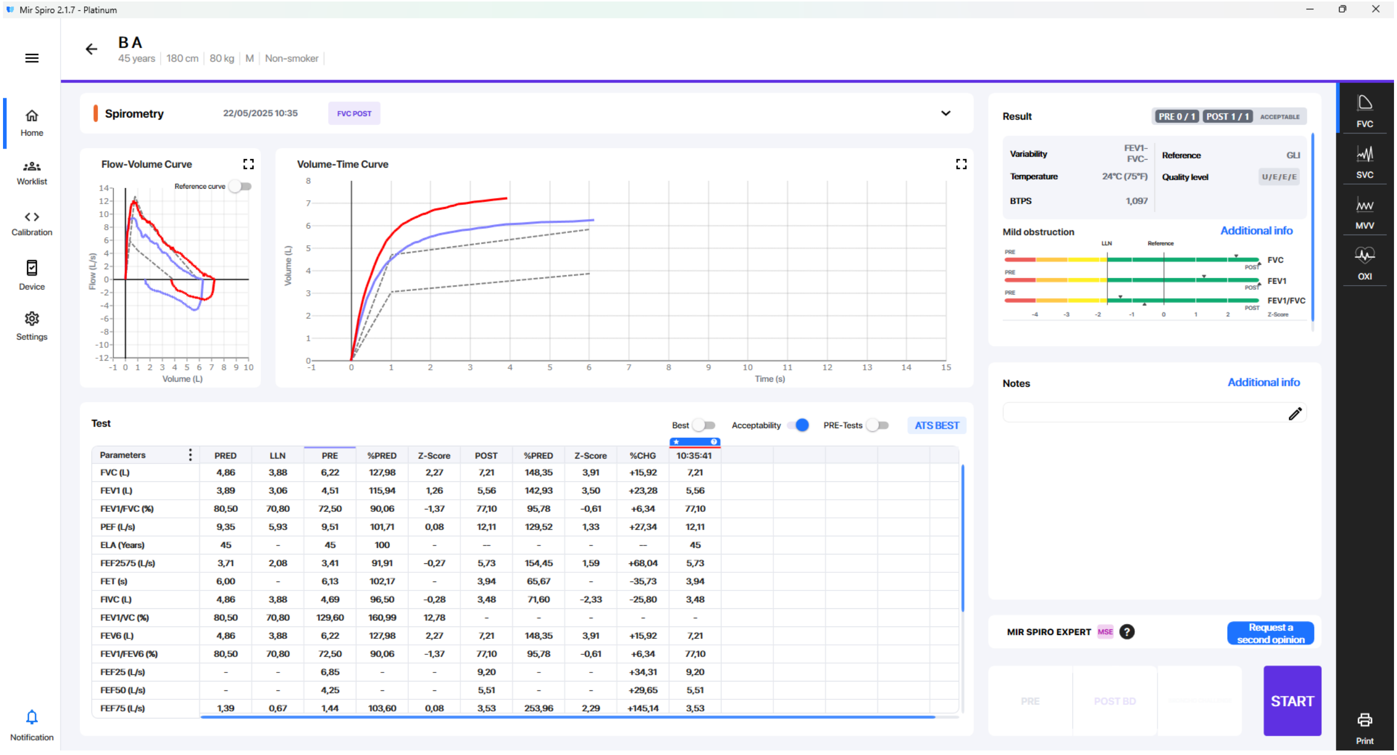
Mild obstruction Additional info

Notes Additional info

MIR SPIRO EXPERT MSE Request a second opinion

Test Best Acceptability PRE-Tests ATS BEST

| Parameters | PRED | LLN | PRE | %PRED | Z-Score | POST | %PRED | Z-Score | %CHG | 10:35:41 |
|---|---|---|---|---|---|---|---|---|---|---|
| FVC (L) | 4,86 | 3,88 | 6,22 | 127,98 | 2,27 | 7,21 | 148,35 | 3,91 | +15,92 | 7,21 |
| FEV1 (L) | 3,89 | 3,06 | 4,51 | 115,94 | 1,26 | 5,56 | 142,93 | 3,50 | +23,28 | 5,56 |
| FEV1/FVC (%) | 80,50 | 70,80 | 72,50 | 90,06 | -1,37 | 77,10 | 95,78 | -0,61 | +6,34 | 77,10 |
| PEF (L/s) | 9,35 | 5,93 | 9,51 | 101,71 | 0,08 | 12,11 | 129,52 | 1,33 | +27,34 | 12,11 |
| ELA (Years) | 45 | - | 45 | 100 | - | -- | - | - | -- | 45 |
| FEF2575 (L/s) | 3,71 | 2,08 | 3,41 | 91,91 | -0,27 | 5,73 | 154,45 | 1,59 | +68,04 | 5,73 |
| FET (s) | 6,00 | - | 6,13 | 102,17 | - | 3,94 | 65,67 | - | -35,73 | 3,94 |
| FIVC (L) | 4,86 | 3,88 | 4,69 | 96,50 | -0,28 | 3,48 | 71,60 | -2,33 | -25,80 | 3,48 |
| FEV1/VC (%) | 80,50 | 70,80 | 129,60 | 160,99 | 12,78 | - | - | - | - | - |
| FEV6 (L) | 4,86 | 3,88 | 6,22 | 127,98 | 2,27 | 7,21 | 148,35 | 3,91 | +15,92 | 7,21 |
| FEV1/FEV6 (%) | 80,50 | 70,80 | 72,50 | 90,06 | -1,37 | 77,10 | 95,78 | -0,61 | +6,34 | 77,10 |
| FEF25 (L/s) | - | - | 6,85 | - | - | 9,20 | - | - | +34,31 | 9,20 |
| FEF50 (L/s) | - | - | 4,25 | - | - | 5,51 | - | - | +29,65 | 5,51 |
| FEF75 (L/s) | 1,39 | 0,67 | 1,44 | 103,60 | 0,08 | 3,53 | 253,96 | 2,29 | +145,14 | 3,53 |



| Variability | FEV1-<br>FVC- | Reference | GLI |
|---|---|---|---|
| Temperature | 24°C (75°F) | Quality level | U/E/E/E |
| BTPS | 1,097 | | |

Test

| Parameters | PRED | LLN | PRE | %PRED | Z-Score | POST | %PRED | Z-Score | %CHG | 10:35:41 |
|---|---|---|---|---|---|---|---|---|---|---|
| FVC (L) | 4,86 | 3,88 | 6,22 | 127,98 | 2,27 | 7,21 | 148,35 | 3,91 | +15,92 | 7,21 |
| FEV1 (L) | 3,89 | 3,06 | 4,51 | 115,94 | 1,26 | 5,56 | 142,93 | 3,50 | +23,28 | 5,56 |
| FEV1/FVC (%) | 80,50 | 70,80 | 72,50 | 90,06 | -1,37 | 77,10 | 95,78 | -0,61 | +6,34 | 77,10 |
| PEF (L/s) | 9,35 | 5,93 | 9,51 | 101,71 | 0,08 | 12,11 | 129,52 | 1,33 | +27,34 | 12,11 |
| ELA (Years) | 45 | - | 45 | 100 | - | -- | - | - | -- | 45 |
| FEF2575 (L/s) | 3,71 | 2,08 | 3,41 | 91,91 | -0,27 | 5,73 | 154,45 | 1,59 | +68,04 | 5,73 |
| FET (s) | 6,00 | - | 6,13 | 102,17 | - | 3,94 | 65,67 | - | -35,73 | 3,94 |
| FIVC (L) | 4,86 | 3,88 | 4,69 | 96,50 | -0,28 | 3,48 | 71,60 | -2,33 | -25,80 | 3,48 |
| FEV1/VC (%) | 80,50 | 70,80 | 129,60 | 160,99 | 12,78 | - | - | - | - | - |
| FEV6 (L) | 4,86 | 3,88 | 6,22 | 127,98 | 2,27 | 7,21 | 148,35 | 3,91 | +15,92 | 7,21 |
| FEV1/FEV6 (%) | 80,50 | 70,80 | 72,50 | 90,06 | -1,37 | 77,10 | 95,78 | -0,61 | +6,34 | 77,10 |
| FEF25 (L/s) | - | - | 6,85 | - | - | 9,20 | - | - | +34,31 | 9,20 |
| FEF50 (L/s) | - | - | 4,25 | - | - | 5,51 | - | - | +29,65 | 5,51 |
| FEF75 (L/s) | 1,39 | 0,67 | 1,44 | 103,60 | 0,08 | 3,53 | 253,96 | 2,29 | +145,14 | 3,53 |

Maintenance activities apply to the devices compatible with the software, and therefore should be carried out in accordance with the maintenance instructions provided for those specific devices.

As for the MIR SUPER SDK, maintenance is relevant only for software developers, and concerns updates to the various software components involved, as explained in the following:

- Application SDKs

With each new version of an SDK, our developer portal automatically notifies all registered integrators, ensuring they are kept up to date with the latest changes. Each SDK release is accompanied by comprehensive documentation, including a list of updates, modifications, and information on compatible devices and operating system versions.

- Web API

When new versions of the Web API are released, communications are sent to integrators to inform them of the changes. A detailed changelog is provided for every new release, and when major changes occur—particularly involving API routes—a dedicated Migration Guide is also made available to assist with the transition.

## 4. Troubleshooting

Follow these instructions to address and resolve issues effectively:

- Consult the troubleshooting section of the Annexes for software developers.
- 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.

## 5. Manufacturer's responsibility and liability

### 5.1 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.

### 5.2 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.

### 5.3 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.

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

### 5.4 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.

### 5.5 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.

# 6. ANNEX A. INSTRUCTION FOR USE - SSDK IOS

## 6.1 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 iOS is only compatible with Spirobank Smart, Smart One and Spirobank II Smart devices.

## 6.2 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)

## 6.3 Integration (How to implement?)

### 6.3.1 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.

### 6.3.2 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 beer 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.

### 6.3.3 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 “multutest mode”. This means that different kind of tests can be started.
At present the tests supported are:

| Test supported | Description | Note |
|---|---|---|
| FVC test | a forced expiratory maneuver | |
| PeakFlow/Fev1 | a forced expiratory maneuver that lasts 1 second | works only with SpirobankSmart with firmware >= 3.0 |
| Flow Monitoring Test | | works only with the new SpirobankSmart Oxi |
| Oximetry test | | works only with the new SpirobankSmart Oxi |
| FVC PLUS Test | a forced expiratory and inspiratory maneuver | works only with ENABLED device: SpirobankSmart with firmware >= 3.1 and Spirobank OXI with firmware >= 1.0 |
| SVC | A slow expiratory and inspiratory maneuver | 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)

Overloads of the “start test” method has been implemented by this framework.
New argument endOfTestTimeout was added in order to set the End of Test timeout (see below: End Of Test Timeout). The new method 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)startTestWithTestType:(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 spirobankSmart equipped with firmware >= 2.7 , ,
This instruction does affects the reading made by the device because different algorithms are used for each turbine type. the StartTest method (whatever is the overload void) soDeviceDidStartTest:(SODevice ctually READY to take
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 mnjhu
(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 |
|---|---|
| pef_cLs (Peakflow cL sec) | FVC-PEAKFLOW/FEV1 |
| fev1_cL (Forced Exp Vol at 1th sec in cL) | FVC-PEAKFLOW/FEV1 |
| quality (acceptability calculation) | FVC-PEAKFLOW/FEV1 |
| fvc_cL (Forced Exp Capacity in cL) | FVC |
| fev1_fvc_pcnt (Fev1% in percentage) | FVC |
| fev6_cL (Forced Exp Volume at 6th sec in cL) | FVC |
| fef2575_cLs (Max mid-expiratory flow in cL sec) | FVC |
| eVol_mL (Extrapolated volume in mL) | PEAKFLOW/FEV1 |
| pefTime_sec (time to reach Peakfow in sec) | PEAKFLOW/FEV1 |
| FVC PLUS TEST | |
| 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_pcnt (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 | |
| 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 |
| HeaxrtRateMax (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 spirobanksmart 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.

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 esier 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 >= 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)

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

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<br>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
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).

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

### 6.3.4 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)
Get an instance of the SODeviceManager Class
Perform a scan and connect OR perform a "direct connection" to a spirometer
Start the test (user must start blowing within the EOT timeout: default =15 sec)
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")
Use didUpdateFlowValue delegate method to show the animated feedback
(also in connection with the SOPatient class's dedicated methods)
Use didUpdateResults to show the result (this method might not be called if the device was not able to calculate the results)
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
Repeat from step 4 (3 times at least)
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)
Get an instance of the SODeviceManager Class
Perform a scan an connect OR perform a "direct connection" to a spirometer
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")
Use a visual feedback to prevent that user start blowing before the app has received the delegate method soDeviceDidStartTest
Use didUpdateFlowValue delegate method to show the animated feedback
(also in connection with the SOPatient class's dedicated methods)
Use didUpdateResults to show the result (this method might not be called if the device was not able to calculate the results)
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).
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
Get an instance of the SODeviceManager Class
Perform a scan an connect OR perform a "direct connection" to a spirometer
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")
Use a visual feedback to prevent that user start blowing before the app has received the delegate method soDeviceDidStartTest
Use didUpdateFlowValue 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 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+)

Get an instance of the SODeviceManager Class
Perform a scan an connect OR perform a "direct connection" to a spirometer
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")
Use a visual feedback to prevent that user start blowing before the app has received the delegate method soDeviceDidStartTest

Use didUpdateVcVolumeTimePoint delegate method to show the animated feedback
Before the ventilatoryProfilePerformed delegate method is called prompt the user to breath normally (tidal breathing / ventilatory profile)
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.
Use ResultsVcUpdated to show the result (this method might not be called if the device was not able to calculate the results)

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:
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
when a significant inhaled volume is detected AND an exhalation has been performed.
when a timeout is expired. A timeout expires in the following conditions:
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
when the user stop blowing for 3 sec
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:
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
when the user stop blowing for 3 sec
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:
when the spirobankSmart detects a volume < 200mL AND a flow <
300mL/s within a timeframe of 2 seconds [here the test is AUTOMATICALLY RESTARTED]
when a significant inhaled volume is detected [here the test is AUTOMATICALLY RESTARTED]
when a timeout is expired. A timeout expires in the following conditions:
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
when the user stop blowing for 3 sec [here the test is
AUTOMATICALLY RESTARTED]
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

OVERVIEW - The SVC Test Guide

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:

evc
ivc
ic
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:
when a timeout is expired. A timeout expires in the following conditions:
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
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:(SOResultsVc *)results;

The SOResultsFVCPlus 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), slowExpInsTime_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 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 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; |

OVERVIEW The FVC Plus Test Guide

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:

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

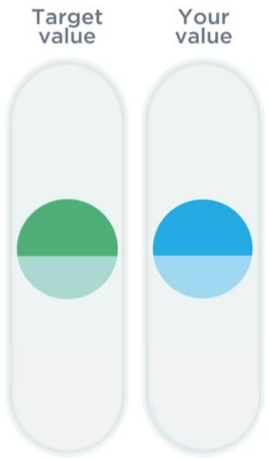
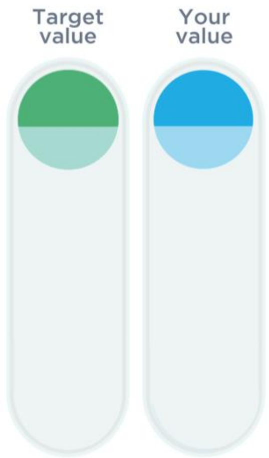
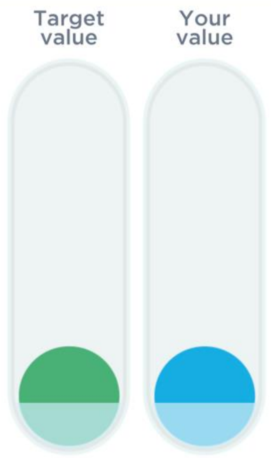
-(void)soDevice:(SODevice *)soDevice didUpdateFvcPlusFlowVolumePoint:(flowVolmePoint *) fvPoint isFirstPackage:(BOOL) isFirstPackage;

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

| | | |
|---|---|---|
|  |  |  |
| 0% | 100 % | -100 % |

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:
when a timeout is expired. A timeout expires in the following conditions:
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
when the user stop blowing for 3 sec
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)eofeIndicator;

```
typedef NS_ENUM(NSInteger, 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:(SOResultsFvcPlus *)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_pcnt (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
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).
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;

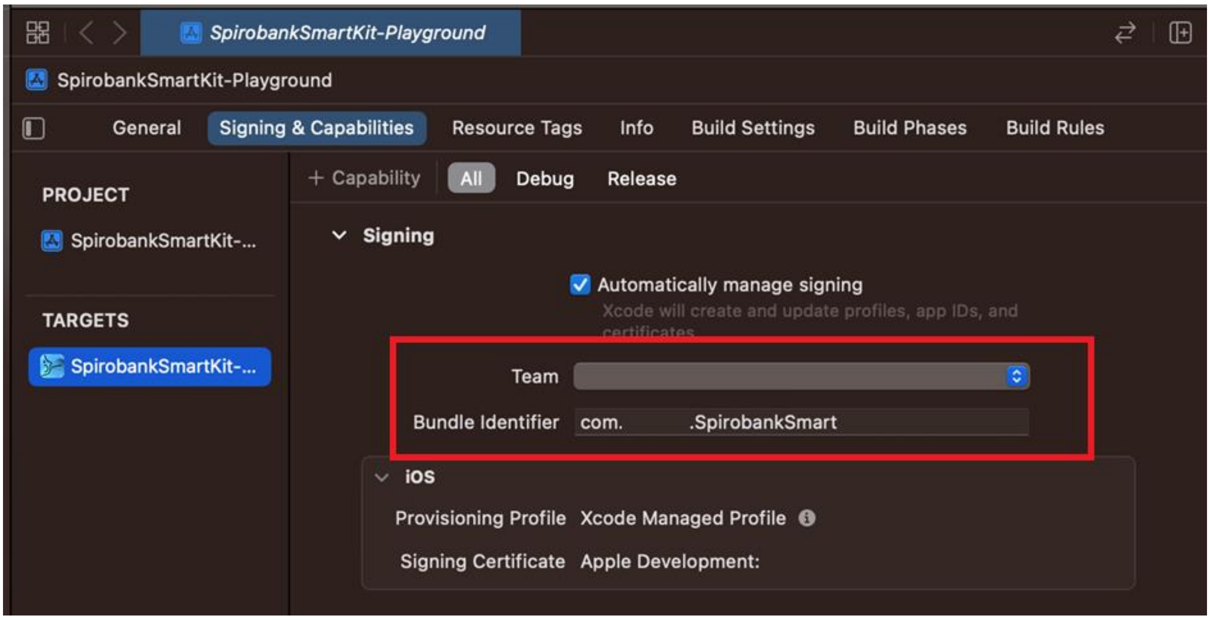
The predicted (in L / Ls, s) , LLN (in L / Ls, s ) and zScore (positive or negative number) values provided (as double) are the following:

| | | |
|---|---|---|
| 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 |

## 6.4 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:

Open the file SpirobankSmartKit-Playground.xcodeproj with Xcode.
In Xode, in the project settings, in the tab "Signing and Capabilities", select your own Apple certificate:



Signing

☑ Automatically manage signing

Xcode will create and update profiles, app IDs, and certificates.

Team

Bundle Identifier com.  .SpirobankSmart

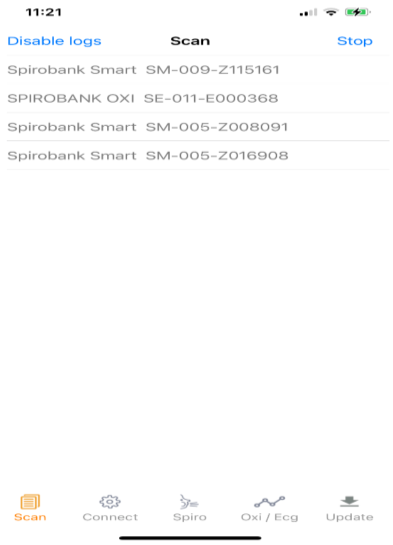
iOS

Provisioning Profile Xcode Managed Profile

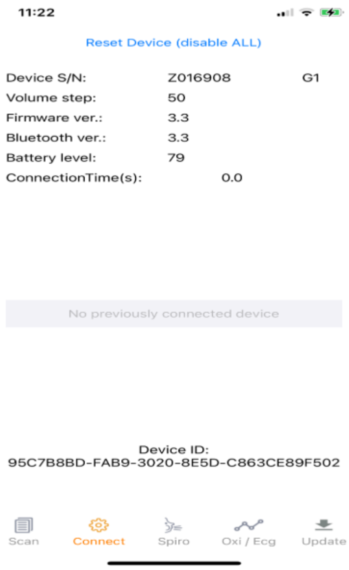
Signing Certificate Apple Development:

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

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:



Spirobank Smart SM-009-Z115161
SPIROBANK OXI SE-011-E000368
Spirobank Smart SM-005-Z008091
Spirobank Smart SM-005-Z016908

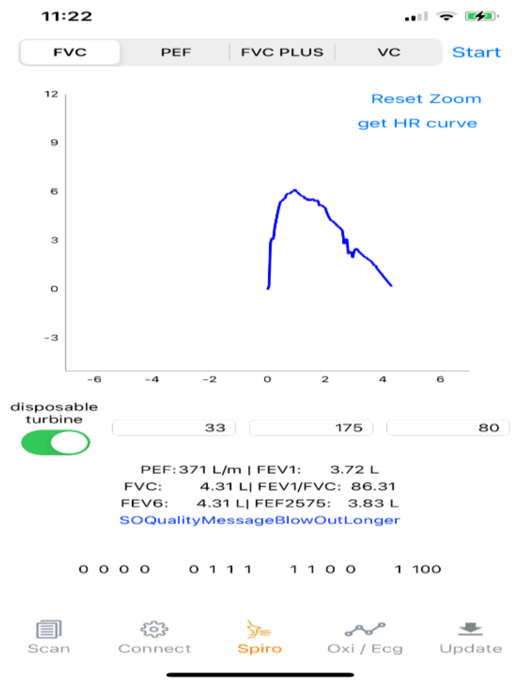


```
Device S/N:        Z016908        G1
Volume step:       50
Firmware ver.:     3.3
Bluetooth ver.:    3.3
Battery level:     79
ConnectionTime(s):     0.0
```

No previously connected device

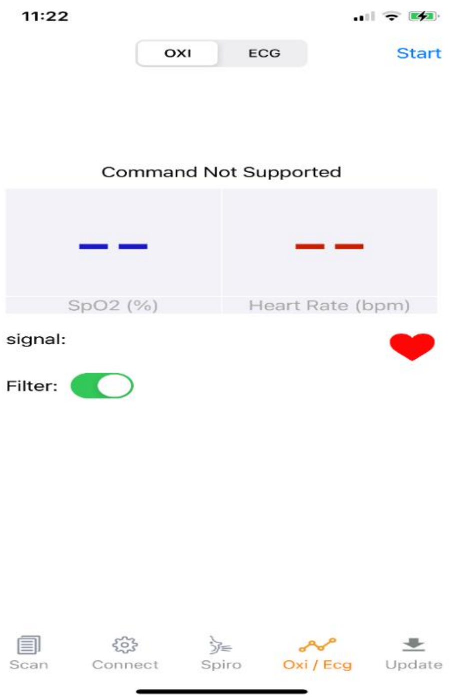
Device ID:
95C7B8BD-FAB9-3020-8E5D-C863CE89F502

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



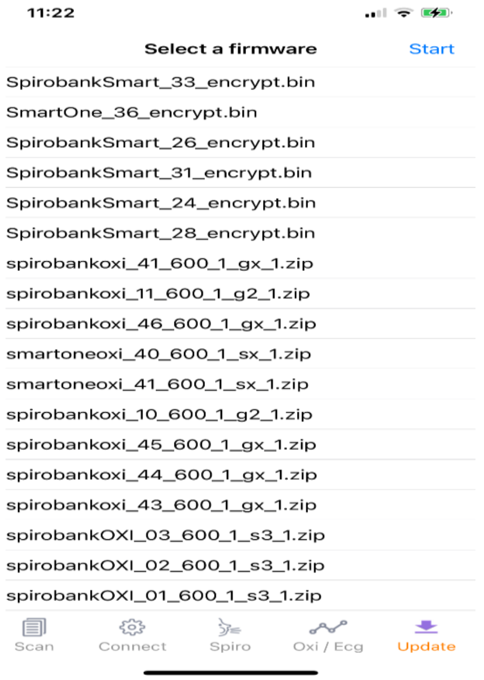
PEF: 371 L/m | FEV1: 3.72 L
FVC: 4.31 L| FEV1/FVC: 86.31
FEV6: 4.31 L| FEF2575: 3.83 L
SOQualityMessageBlowOutLonger

0 0 0 0 0 1 1 1 1 1 0 0 1 100



Command Not Supported

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



Select a firmware

SpirobankSmart_33_encrypt.bin
SmartOne_36_encrypt.bin
SpirobankSmart_26_encrypt.bin
SpirobankSmart_31_encrypt.bin
SpirobankSmart_24_encrypt.bin
SpirobankSmart_28_encrypt.bin
spirobankoxi_41_600_1_gx_1.zip
spirobankoxi_11_600_1_g2_1.zip
spirobankoxi_46_600_1_gx_1.zip
smartoneoxi_40_600_1_sx_1.zip
smartoneoxi_41_600_1_sx_1.zip
spirobankoxi_10_600_1_g2_1.zip
spirobankoxi_45_600_1_gx_1.zip
spirobankoxi_44_600_1_gx_1.zip
spirobankoxi_43_600_1_gx_1.zip
spirobankOXI_03_600_1_s3_1.zip
spirobankOXI_02_600_1_s3_1.zip
spirobankOXI_01_600_1_s3_1.zip

Additional Resources
MIR Website: https://www.spirometry.com

## 6.5 Troubleshooting

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:

Bundle Identifier 'com.spirometry.SpirobankSmart has already been used.
Solution: Modify the Bundle Identifier in your project settings to make it unique.

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.

# 7. ANNEX B. INSTRUCTION FOR USE - MIR SUPER SDK WINDOWS

## 7.1 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.

## 7.2 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

## 7.3 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."

## 7.4 Sample Demo Application

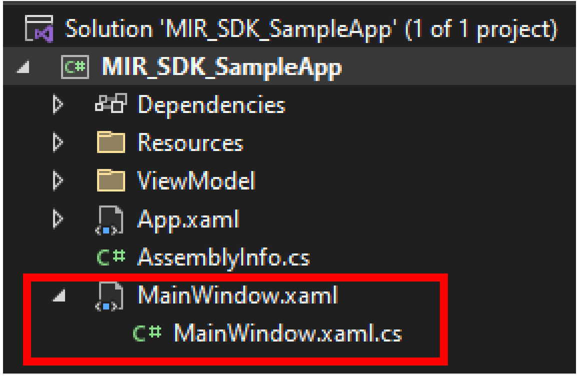
The "mir_sdk_windows_<version>" archive contains a subfolder named " MIR SDK <version>\MIR SDK - SampleApp ".

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

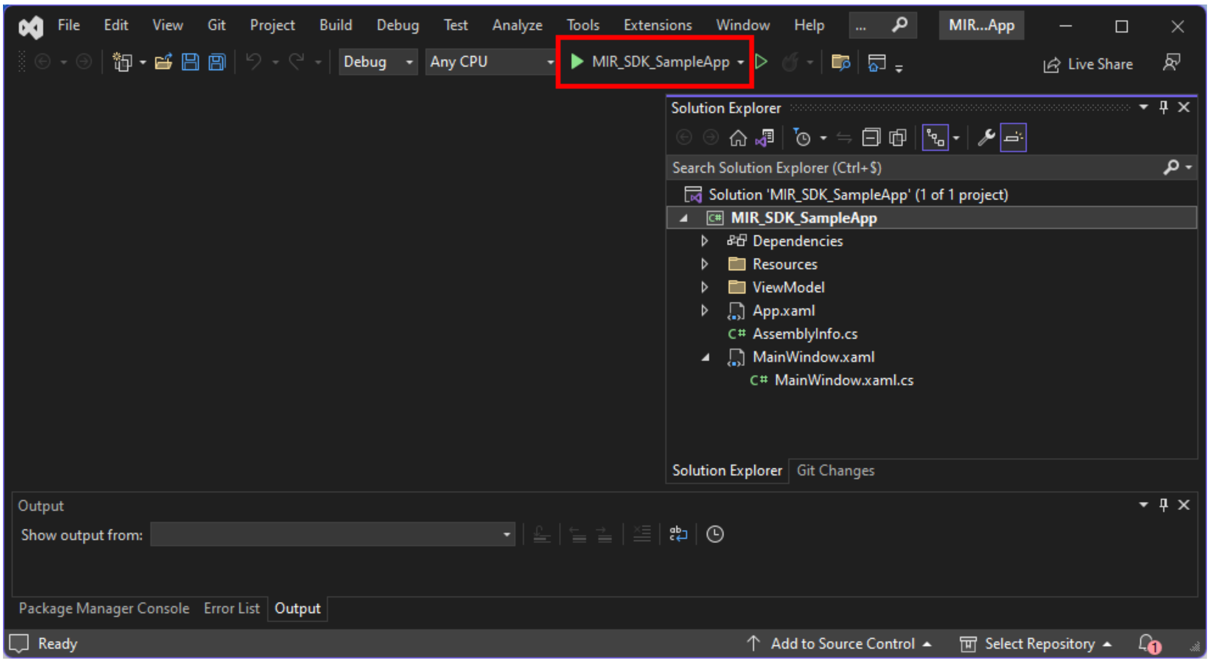
Open this file with Microsoft Visual Studio. Then, ensure you have .NET Runtime installed on your machine.

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

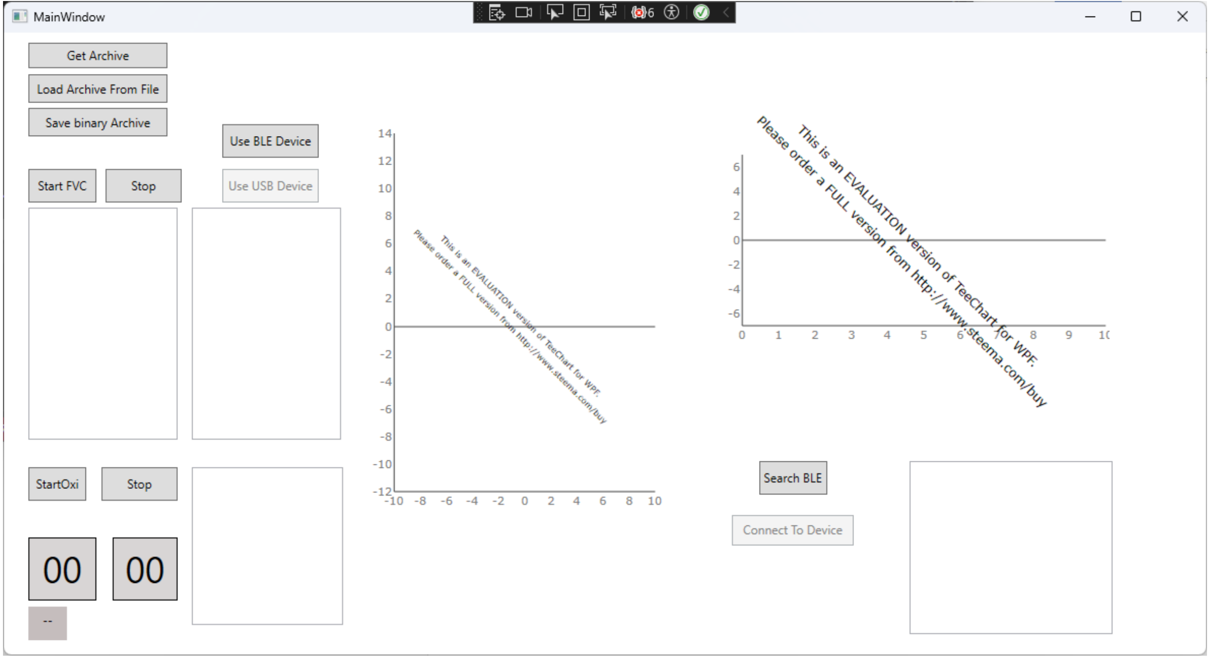
This class features a WPF graphical interface, allowing you to see the function called behind each button:



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.

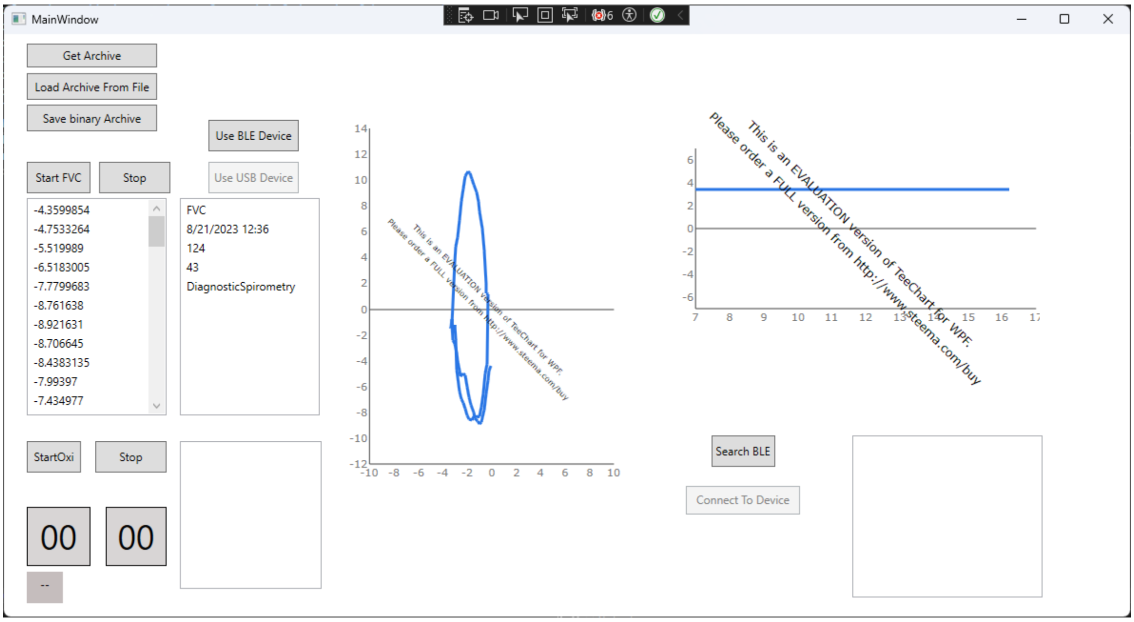
Labels 00 and 00: Values of current BPM and SpO2 when Oximetry test is running.

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

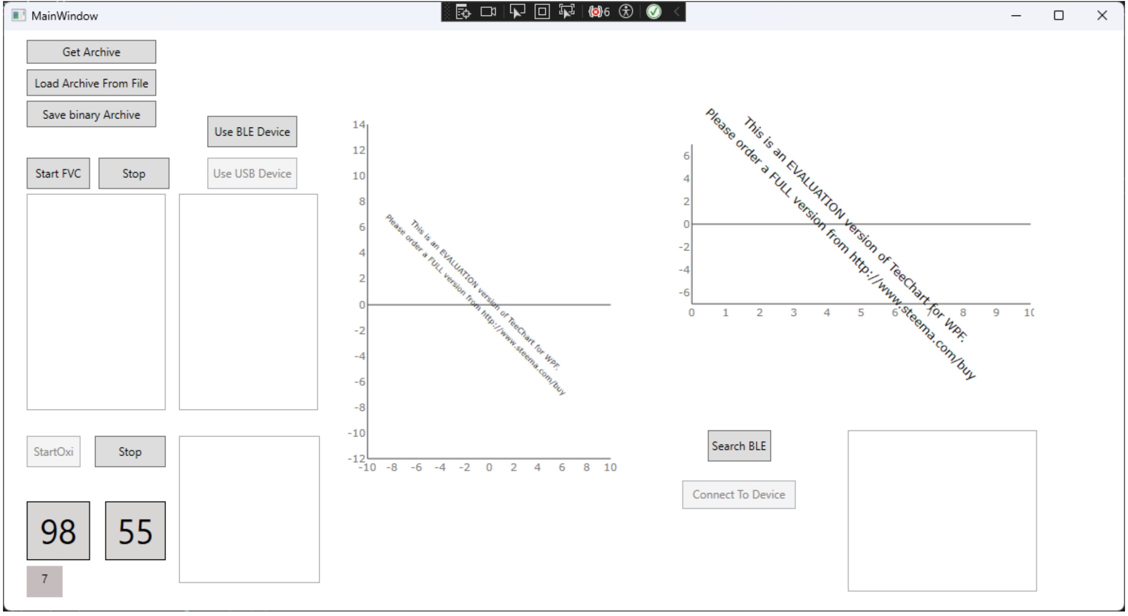


-4.3599854
-4.7533264
-5.519989
-6.5183005
-7.7799683
-8.761638
-8.921631
-8.706645
-8.4383135
-7.99397
-7.434977

FVC
8/21/2023 12:36
124
43
DiagnosticSpirometry

00 00
--

Example of screen during an Oximetry session

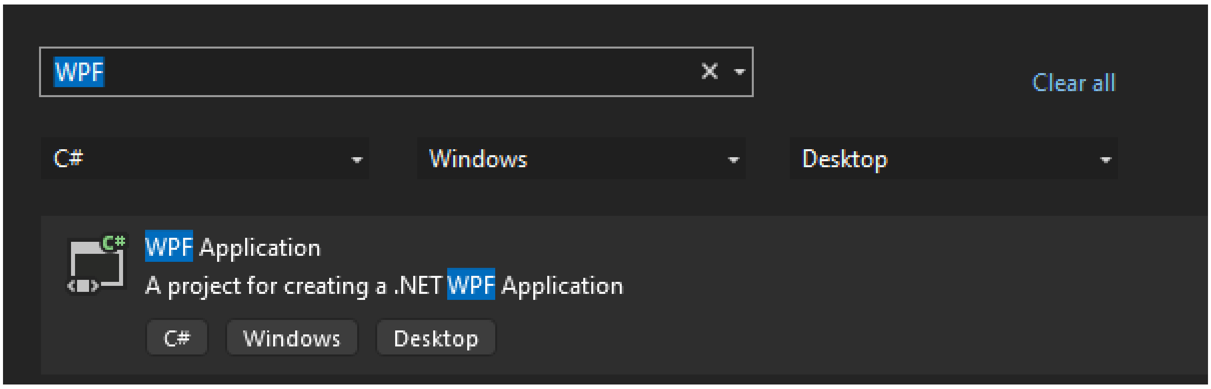


98 55
7

Integration into a Microsoft Visual Studio project

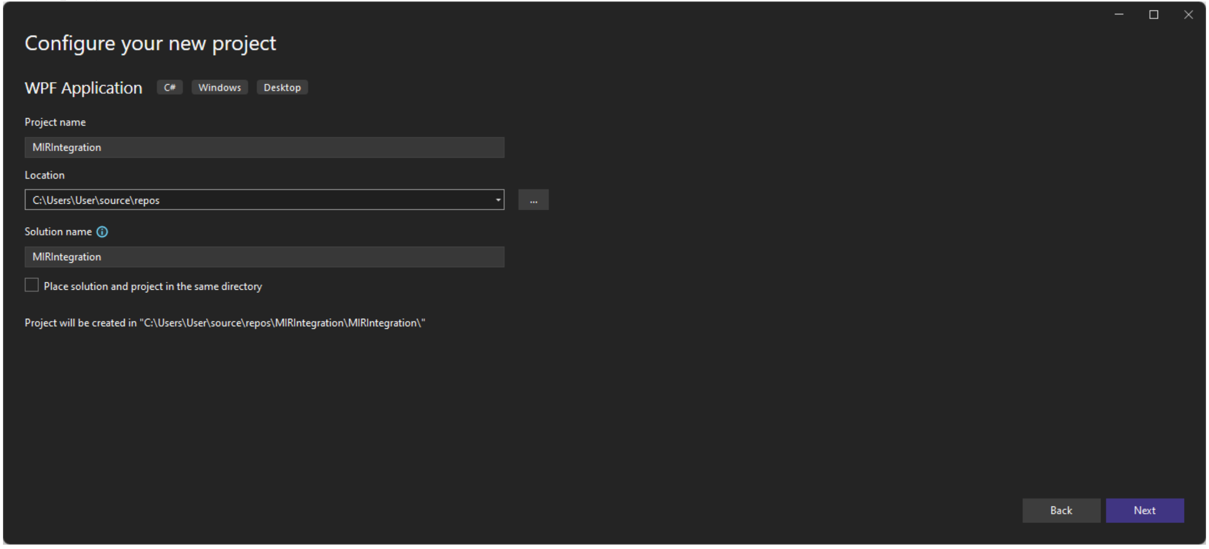
For this example, we'll create a blank WPF project with a simple button that will call the SDK to retrieve the file "MIRXFile".

Open Microsoft Visual Studio:


WPF

WPF Application
A project for creating a .NET WPF Application

Enter the project name and directory:


Configure your new project

WPF Application

Project name: MIRIntegration

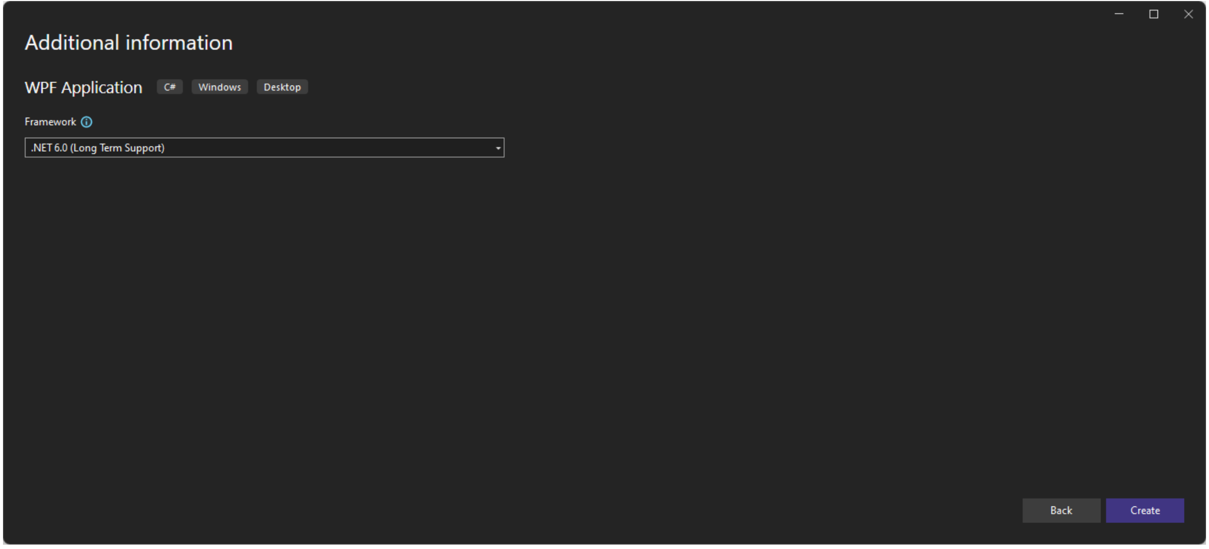
Location: C:\Users\User\source\repos

Solution name: MIRIntegration

□ Place solution and project in the same directory

Project will be created in "C:\Users\User\source\repos\MIRIntegration\MIRIntegration\"

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


Additional information

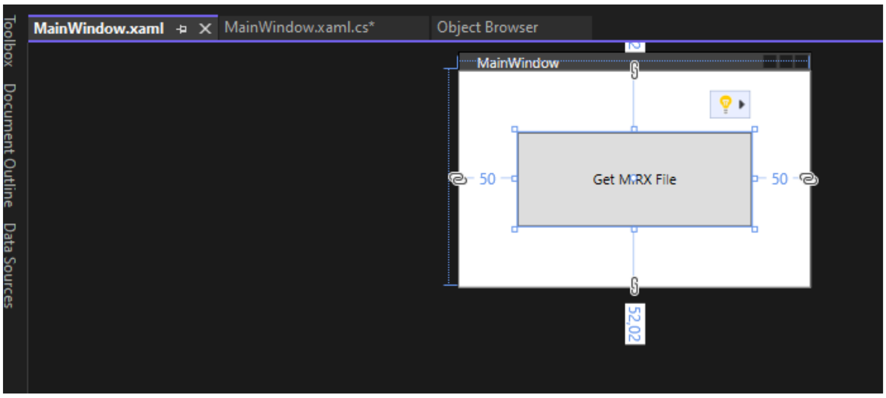
WPF Application

Framework: .NET 6.0 (Long Term Support)

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

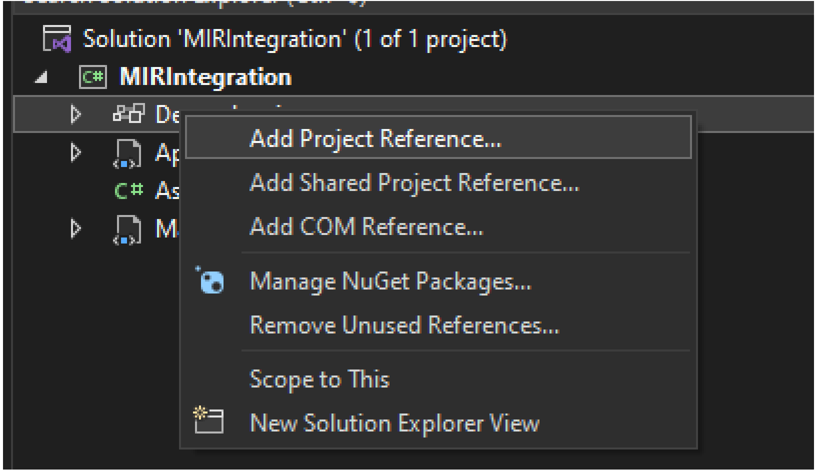
```
<Grid>
    <Button
        x:Name="btnGetMirXFile"
        Width="200"
         Height="80"
         Click="btnGetMirXFile_Click">
         Get MIRX File
    </Button>
```

```
</Grid>
```

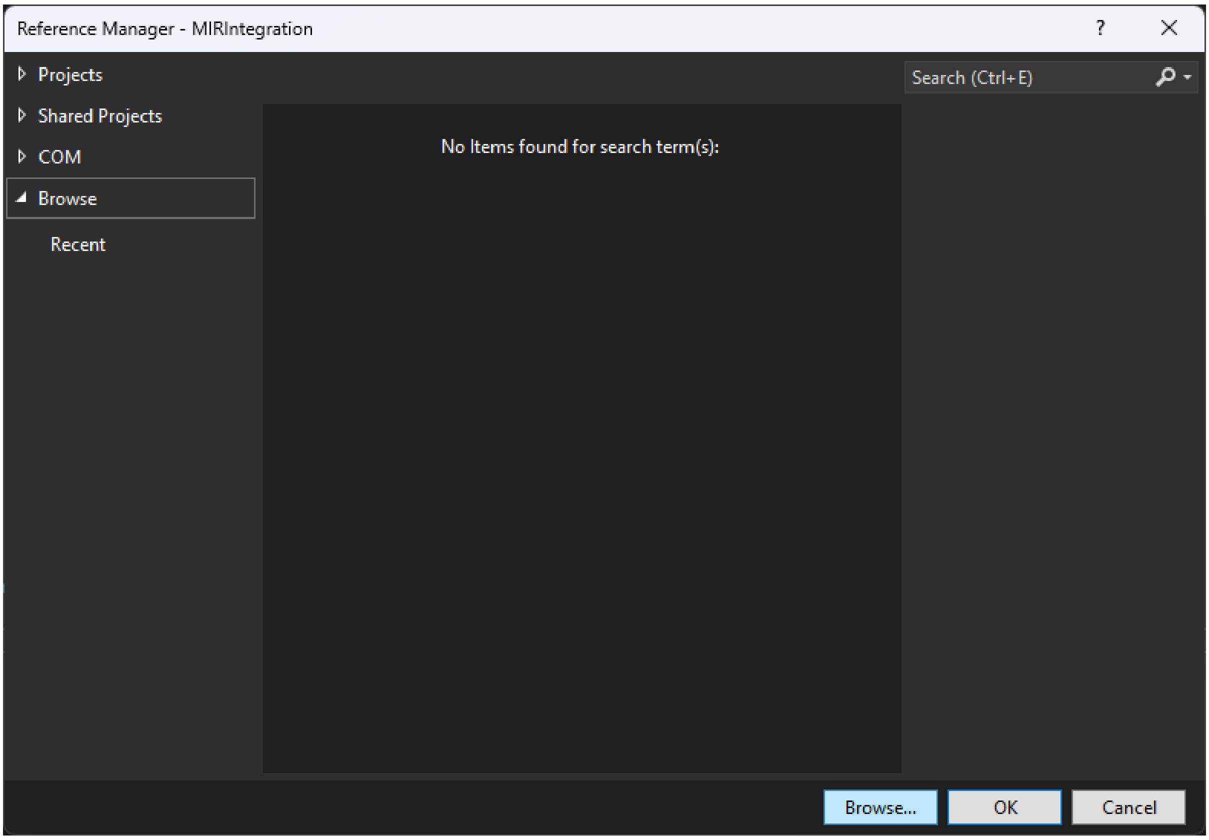


It is now necessary to add references to the SDK.

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

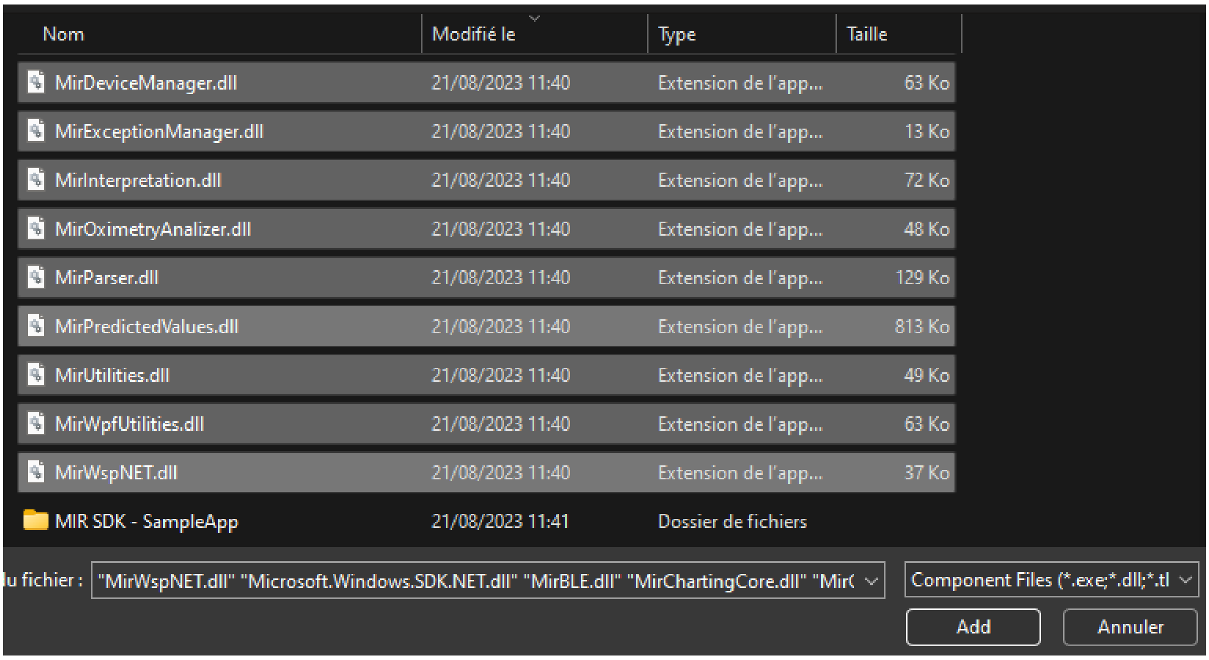

Solution 'MIRIntegration' (1 of 1 project)
MIRIntegration
De
Ap
As
M
Add Project Reference...
Add Shared Project Reference...
Add COM Reference...
Manage NuGet Packages...
Remove Unused References...
Scope to This
New Solution Explorer View

Press "Browse":



No Items found for search term(s):

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



| Nom | Modifié le | Type | Taille |
|---|---|---|---|
| MirDeviceManager.dll | 21/08/2023 11:40 | Extension de l'app... | 63 Ko |
| MirExceptionManager.dll | 21/08/2023 11:40 | Extension de l'app... | 13 Ko |
| MirInterpretation.dll | 21/08/2023 11:40 | Extension de l'app... | 72 Ko |
| MirOximetryAnalizer.dll | 21/08/2023 11:40 | Extension de l'app... | 48 Ko |
| MirParser.dll | 21/08/2023 11:40 | Extension de l'app... | 129 Ko |
| MirPredictedValues.dll | 21/08/2023 11:40 | Extension de l'app... | 813 Ko |
| MirUtilities.dll | 21/08/2023 11:40 | Extension de l'app... | 49 Ko |
| MirWpfUtilities.dll | 21/08/2023 11:40 | Extension de l'app... | 63 Ko |
| MirWspNET.dll | 21/08/2023 11:40 | Extension de l'app... | 37 Ko |
| MIR SDK - SampleApp | 21/08/2023 11:41 | Dossier de fichiers | |

u fichier : "MirWspNET.dll" "Microsoft.Windows.SDK.NET.dll" "MirBLE.dll" "MirChartingCore.dll" "Mir(

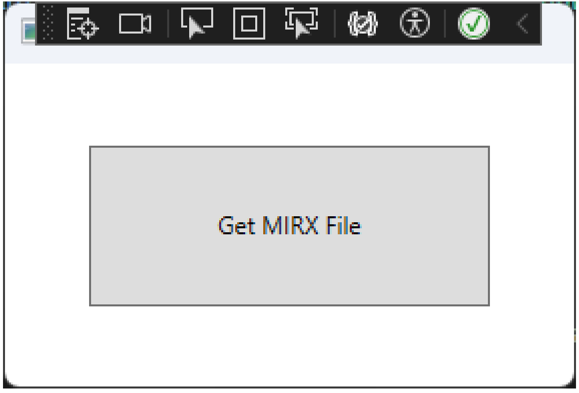
Component Files (*.exe;*.dll;*.tl

Click on “Add” and then “OK”.

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();
        }
    }
}
```

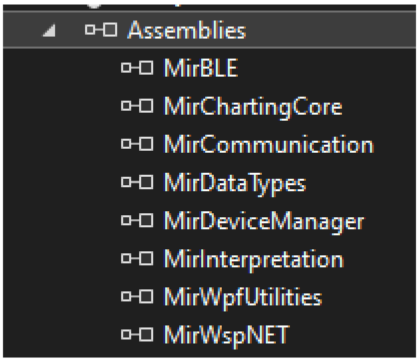
Launch the application and conduct the test.



You can now use the full SDK in your project.

### 7.5 Main features

e 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)

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.

USB connection

Create an instance of the USBManager class to open the connection

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

Create an instance of the MirDevice class to connect the device

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

BLE connection

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

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

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);
```

Start discovery method:

```
BleManager.StartDiscovery()
```

To force discovery interruption call BleManager.StopDiscovery() method.
Connect to discovered device.

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

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

Start downloading the tests from the device:

```
myDevice.GetArchive(MirInterpretationConfiguration.GenerateDefault(PredictedValuesProvider.Knudson_Kn
udson));
```

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

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)
{
   label1.Text = "Start Archive .......";
}
```

Interpretation of data downloaded from device complete:

```
myDevice.OnEndParsing += new EventHandler(OnArchiveParsed);
void OnArchiveParsed(object sender, EventArgs e)
{
  label1.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)
{
   label1.Content = "Parsing Complete"
}
```

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

Connect to the device

USB

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

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

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

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

BLE

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)
```

Conduct the FVC test

Start the FVC test

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

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);
}
```

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);
}
```

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);
}
```

End the test

```
myDevice.StopTest()
```

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

Connect to the device
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();
```

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)
```

Conduct the VC test
Start the FVC test:

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

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);
}
```

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");
}
```

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

Select the .mir or .mirx file
Create the subject OpenFileDialog:

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

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

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

View the file selection window through the ShowDialog function:

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

Interpret the content of the selected file

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);
```

Generate the virtual MIR device:

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

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)
{
    label1.Content = "Parsing Complete"
}
```

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

Connect to the device
USB

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()

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)

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

Start downloading the tests from the device:

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

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"
}
```

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

Connect to the device
USB
Create an instance of the USBManager class to open the connection:

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

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

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

Read the content of the .tsk file

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

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);
}
```

### 7.6 Additional Resources

MIR Website: https://www.spirometry.com

### 7.7 Troubleshooting

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.

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.

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.

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.

# 8. ANNEX C. INSTRUCTION FOR USE - MIR SUPER SDK WEB (CLOUD/API)

## 8.1 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} |

## 8.2 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").

## 8.3 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:

Simple Authentication: By sending a username and password.

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": "ey...", // 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".

Choose a mode to receive the OTP:

Call the /otp/mode route with the payload:

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

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": "ey...",
 "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.

## 8.4 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 application) 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

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: print
Request type: POST
Input data: Session object
Output data: Base64 of the PDF file generated
Description: Return the PDF of the given session (in base64)
/v1/print/{impressionFormat}/dicom
Scope: print
Request type: POST
Input data: Session object
Output data: Base64 of the DICOM file generated
Description: Return the DICOM PDF of the given session (in base64)

## 8.5 Troubleshooting

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.
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.
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.
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.
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.
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.
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.
Persistent Issues Error: Issues remain despite checks.
Solution: Contact our team.

# 9. ANNEX D. INSTRUCTION FOR USE - MIR SUPER SDK MACOS

## 9.1 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.

## 9.2 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.

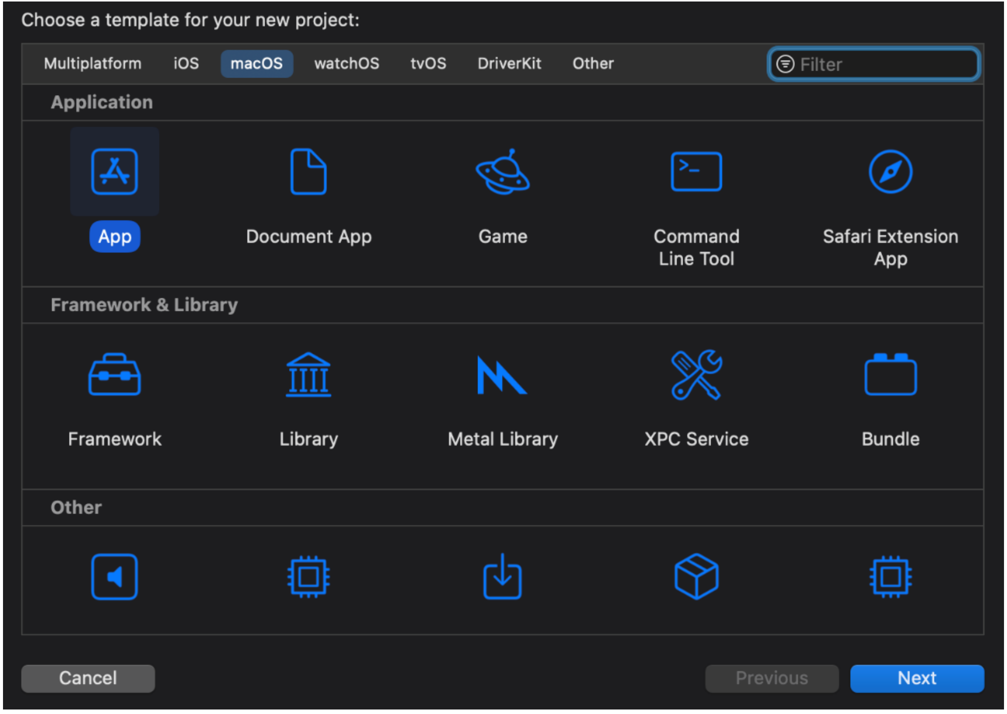
## 9.3 How to implement

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

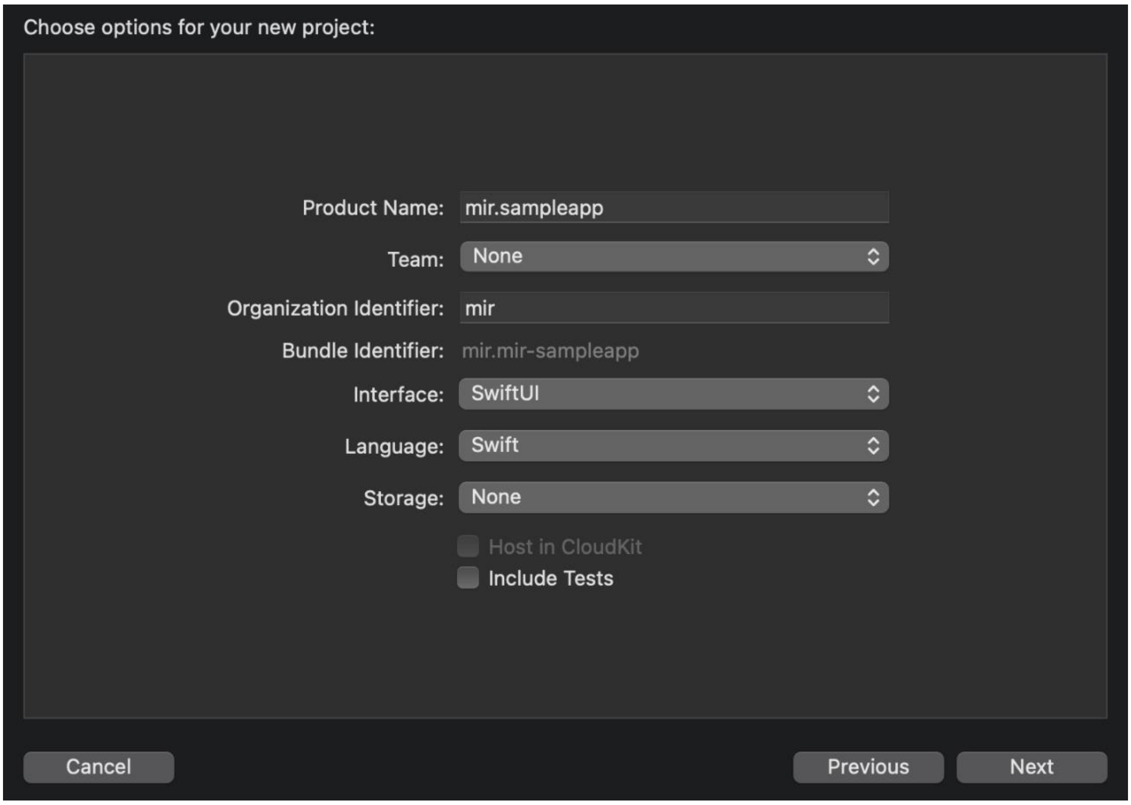
Integration into a XCode project

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

Open Xcode and create new project, choose MacOS and App:



Enter the product name, the organization identifier, the interface and the language:



Choose options for your new project:

Product Name: mir.sampleapp

Team: None

Organization Identifier: mir

Bundle Identifier: mir.mir-sampleapp

Interface: SwiftUI

Language: Swift

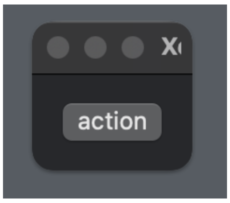
Storage: None

□ Host in CloudKit

□ Include Tests

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()
    }
}
```

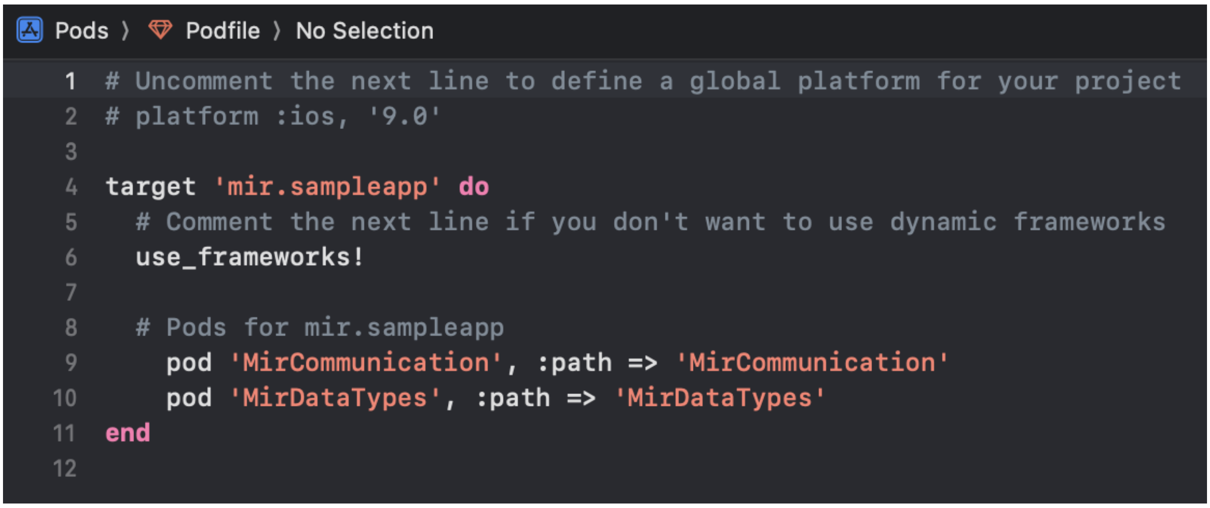


It is now necessary to add references to the SSDK, for this we'll use Cocoapods.
You can find how to install Cocoapods here.

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

```
$ pod init
```

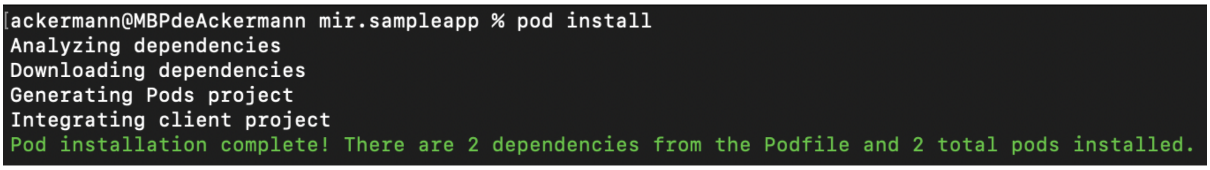
After this, you'll have a file named Podfile, this is where you'll put all your dependencies, for this example we'll use MirCommunication.



```
# Uncomment the next line to define a global platform for your project
# platform :ios, '9.0'

target 'mir.sampleapp' do
  # Comment the next line if you don't want to use dynamic frameworks
  use_frameworks!

  # Pods for mir.sampleapp
    pod 'MirCommunication', :path => 'MirCommunication'
    pod 'MirDataTypes', :path => 'MirDataTypes'
end

```

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

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

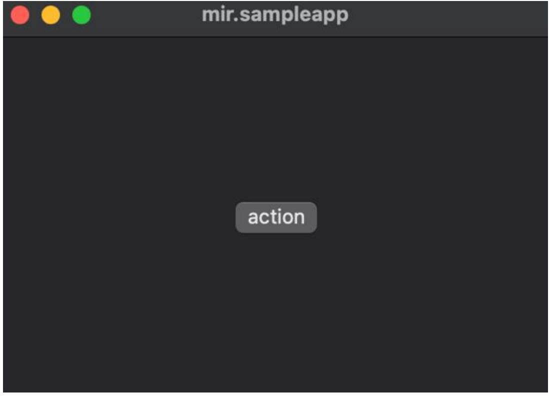


```
[ackermann@MBPdeAckermann mir.sampleapp % pod install
Analyzing dependencies
Downloading dependencies
Generating Pods project
Integrating client project
Pod installation complete! There are 2 dependencies from the Podfile and 2 total pods installed.
```

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()
}
```

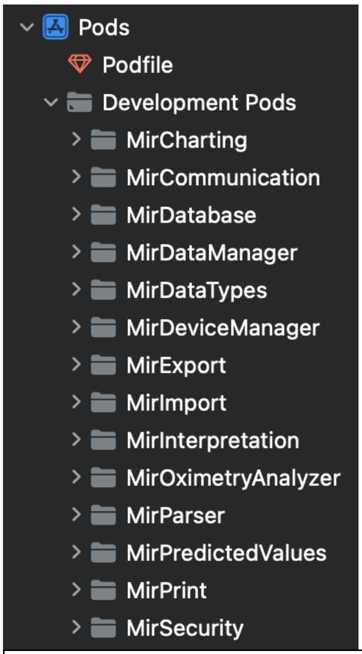
Launch the application and conduct the test.



You can now use the full SDK in your project.

## 9.4 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:



```
Pods
  Podfile
  Development Pods
    MirCharting
    MirCommunication
    MirDatabase
    MirDataManager
    MirDataTypes
    MirDeviceManager
    MirExport
    MirImport
    MirInterpretation
    MirOximetryAnalyzer
    MirParser
    MirPredictedValues
    MirPrint
    MirSecurity
```

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.

USB connection

First find the USB devices by calling the shared instance of MirDeviceManager

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

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)
}
```

You can check discovered devices in this variable.

```
MirDeviceManager.shared.UsbDiscoveredDevices
```

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.

BLE connection

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

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

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]
}
```

You can check discovered devices in this variable.

```
MirDeviceManager.shared.bluetoothDiscoveredDevices
```

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

Connect to the device.
See Connecting to a device.
Conduct the FVC test.
Once you have connected your device, you can get the connected device and start the FVC test.

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

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

```
[…], completion: {
    [weak self] (trial) in

    if let trial = trial {
        self?.trialReceived(trial)
    }
}
```

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.
End the test.

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

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

Connect to the device.
See Connecting to a device.
Conduct the VC test.
Once you have connected your device, you can get the connected device and start the VC test.

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

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

```
[…], completion: {
    [weak self] (trial) in

    if let trial = trial {
        self?.trialReceived(trial)
    }
}
```

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.

End the test.

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

Retrieving an archive and generating the .mirx file

Connect to the device.
See Connecting to a device.
Download the data (not interpreted) by the connected device and generate the .mirx file
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.
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

Connect to the device.
See Connecting to a device.
2) Select the .tsk file to upgrade the firmware of the MIR devices
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)
                        }
                    }
                }
            }
        }
```

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:
        […]
```

## 9.5 Additional Resources

MIR Website: https://www.spirometry.com

Annex E. Instruction for Use - SSDK Android

## 9.6 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.

## 9.7 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)

## 9.8 Integration (How to implement?)

### 9.8.1 Spirobank Smart SDK Android Guide - Import and configure the SDK Module

To import the SDK Module inside your project just take the following steps:
Open your Android Studio Project and go to menu “File” -> “New” -> “New Module…”
Select “Import .JAR/.ARR Package” and click next
Select “spirobanksmartsdk.aar” file and click finish
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.

### 9.8.2 Perform a scan (discovery)

With an instance of deviceManager the client app may call the startDiscovery method.
The discovery will retrieves 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.

### 9.8.3 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 isDataValid) to pass the values measured and oximetryPlethysmographicValuesUpdated (double ppgSignal,int minY,int maxY) to pass pletismographic 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_pcnt (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 Peakfow in sec) | PefFev1 |
| spO2Mean (%) | Oximetry |
| spO2Max (%) | Oximetry |
| spO2Min (%) | Oximetry |
| HeartRateMean (bpm) | Oximetry |
| HeaxrtRateMax (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 |
|---|---|
| getAdvertisementDataName() | Bluetooth device name |
| getAddress() | Bluetooth device address |
| getName() | MIR product name |
| getProtocol() | MIR communication protocol |
| getSerialNumber() | MIR device serial number |

How the test is stopped/restarted in the "multitest mode" environment

The test is stopped in two ways:
when the stopTest method is invoked
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).
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 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)

### 9.8.4 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.

### 9.8.5 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.

### 9.8.6 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.

### 9.8.7 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
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.

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.

### 9.8.8 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:

| Device supported | Firmware supported |
|---|---|
| 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 sofwareUpdateProgress 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.

### 9.8.9 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)
Get an instance of the DeviceManager Class
Perform a scan an connect OR perform a "direct connection" to a spirometer
Add your PefFev1DeviceCallback 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 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.

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 Patient class's dedicated methods)
Use resultsUpdated to show the result (this method might not be called if the device was not able to calculate the results)
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
Repeat from step 4 (3 times at least)
send the stopTest command to quit from test and wait for the testStopped delegate method to be called.
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)
Get an instance of the DeviceManager Class
Perform a scan an connect OR perform a "direct connection" to a spirometer
Add your PefFev1DeviceCallback 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 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.

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 Patient class's dedicated methods)
Use resultsUpdated to show the result (this method might not be called if the device was not able to calculate the results)
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).
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 an 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.

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 an 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.

Use a visual feedback to prevent that user start blowing before the app has received the delegate method testStarted
Use volumeUpdated delegate method to show the animated feedback
When the ventilatoryProfilePerformed callback is called, patient's ventilatory profile is acquired
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 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:
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
when a significant inhaled volume is detected AND an exhalation has been performed.
when a timeout is expired
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
when the user stops blowing for 3 sec
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:
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
when the user stops blowing for 3 sec
when the user keeps on blowing for 60 seconds and no plateau has been reached

**PEAKFLOW/FEV1 test**
The PEAKFLOW/FEV1 maneuver AUTOMATICALLY ends:
when the device detects a volume < 200mL AND a flow < 300mL/s within a timeframe of 2 seconds (here the test is AUTOMATICALLY RESTARTED)
when a significant inhaled volume is detected (here the test is AUTOMATICALLY RESTARTED)
when a timeout is expired

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
when the user stop blowing for 3 sec (here the test is AUTOMATICALLY RESTARTED)
when the user keeps on blowing for 60 seconds AND none of the previous condition has been met

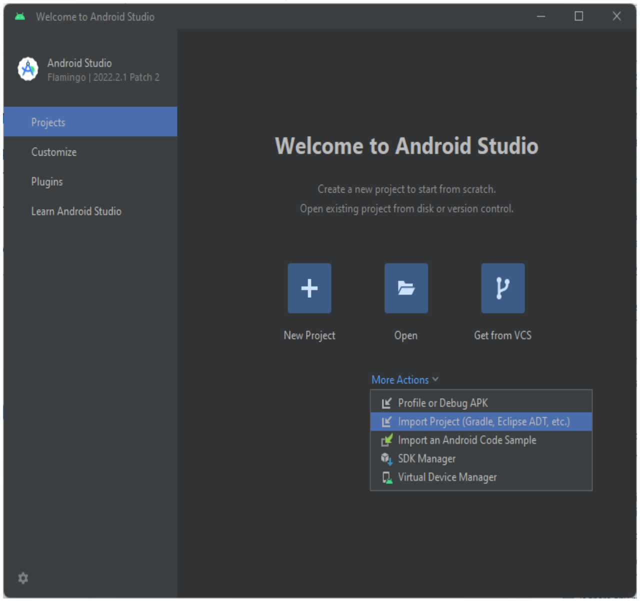
To have an acceptable PEAKFLOW/FEV1 maneuver, the exhalation must last no less than 1 seconds

### VC TEST

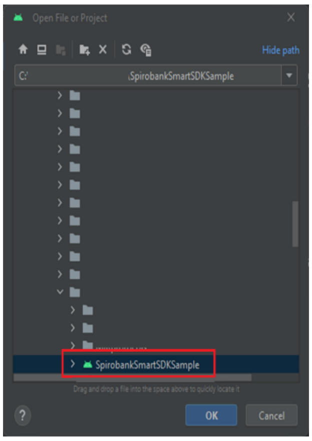
The VC test:
when the user has not blown at all for 15-120 (according to the EndOfTestTimeOut parameter passed to the “startTest” command) seconds since the test was started
when the user stops blowing for 3 sec

## 9.9 Sample Demo Application

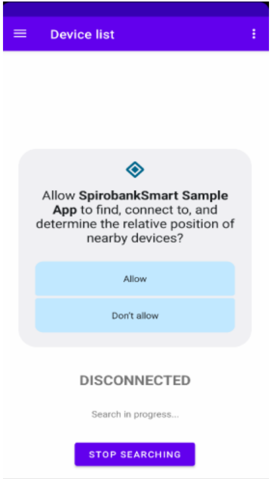
The archive contains a subfolder named " SpirobankSmartSDKSample".
This folder contains an Android studio project.
To use it, open Android Studio and follow the instructions below:
On the main screen, select “More Actions” and select “Import Project (Gradle, Eclipse ADT, etc.)


Welcome to Android Studio
Android Studio
Flamingo | 2022.2.1 Patch 2
Projects
Customize
Plugins
Learn Android Studio
Welcome to Android Studio
Create a new project to start from scratch.
Open existing project from disk or version control.
New Project
Open
Get from VCS
More Actions
Profile or Debug APK
Import Project (Gradle, Eclipse ADT, etc.)
Import an Android Code Sample
SDK Manager
Virtual Device Manager

Select the “SpirobankSmartSDKSample” directory which should be symbolized with the Android icon.


Open File or Project
Hide path
C:/ SpirobankSmartSDKSample
SpirobankSmartSDKSample
Drag and drop a file into the space above to quickly locate it
OK
Cancel

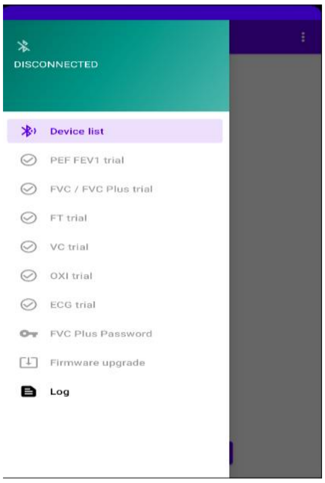
Build the App and start it on a device.
The App allows you to scan the MIR Bluetooth Low Energy devices and to perform different types of tests:



Allow **SpirobankSmart Sample App** to find, connect to, and determine the relative position of nearby devices?

DISCONNECTED

Search in progress...



## 9.10 Additional Resources

MIR Website: https://www.spirometry.com

## 9.11 Troubleshooting

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.

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'
}
```

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.
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.
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.
INSTALL_FAILED_VERSION_DOWNGRADE
Solution: uninstall the existing application from the device or emulator and try installing again.
Emulator: PANIC: Missing emulator engine program for 'x86' CPU.
Solution: Ensure the necessary Android Emulator components are properly installed via the SDK Manager.
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.
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.