Intel® RealSense™ SDK 2015 R4
Developer Guide

API VERSION 6.0
# Table of Contents

- Legal Information .................................................. 4
- Introducing the SDK ................................................. 6
- Getting Started with the SDK ........................................ 7
  - Installing the SDK .................................................. 8
  - Setting Up the Camera ............................................ 9
  - Configuring Development Environment ...................... 11
    - Configuring C++ Development Environment .............. 12
      - Importing the SDK Property Sheets ....................... 13
      - Configuring Project Settings ................................. 14
    - Configuring C# Development Environment ............... 16
      - Customize the C# libraries .................................. 17
    - Configuring Unity Development Environment ............ 18
      - Customizing the Unity Libraries ......................... 19
    - Configuring Processing Development Environment .... 20
    - Configuring Java Development Environment .............. 21
    - Configuring JavaScript Development Environment ...... 22
- Deploying Your Application ......................................... 23
  - Packaging Supporting Libraries ................................ 24
  - Installing SDK runtimes ........................................... 25
    - Speech Runtime and Language Packs ....................... 27
    - Uninstalling SDK runtimes .................................... 28
    - Installer Options .................................................. 29
    - Installer Feature Code ........................................... 31
    - Installer Status Code ............................................ 33
  - Deploying JavaScript Applications ............................... 34
    - Designing the Landing Page .................................... 35
    - Preparing the WebApp Runtime ............................... 37
    - Hosting Your Application ...................................... 39
- Related Topics .......................................................... 40
  - SDK Runtime Upgrade Policy .................................... 41
  - Checking SDK, Camera Driver, and Algorithm Versions .... 42
  - Algorithm Operating Ranges .................................... 45
  - Working with Multiple Modalities ............................... 47
    - Sharing Device Configuration .................................. 58
  - Coexisting with Other Applications ............................ 64
    - Sharing Device Properties ..................................... 65
    - Sharing Platform Computations .............................. 67
  - Working with Screen Rotation ................................. 68
  - Managing Power and Performance ............................. 70
  - Working with Platform Analyzer ............................... 72
    - Setting Up an Application for Analysis .................... 74
Configuring and Running the Analysis

Viewing Platform Analysis Data

Advanced Tracking Operations

Privacy Notice Guidelines
1 Legal Information

By using this document, in addition to any agreements you have with Intel, you accept the terms set forth below.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

Information in this document is provided in connection with Intel products. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. Except as provided in Intel's Terms and Conditions of Sale for such products, Intel assumes no liability whatsoever and Intel disclaims any express or implied warranty, relating to sale and/or use of Intel products including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright or other intellectual property right.

Unless otherwise agreed in writing by Intel, the Intel products are not designed nor intended for any application in which the failure of the Intel product could create a situation where personal injury or death may occur.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "Reserved" or "Undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or by visiting Intel's web site http://www.intel.com.

Any software source code reprinted in this document is furnished under a software license and may only be used or copied in accordance with the terms of that license.

Intel, the Intel logo, Intel RealSense, Intel Core and Iris are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
Microsoft, Windows, and the Windows logo are trademarks, or registered trademarks of Microsoft Corporation in the United States and/or other countries.

Java is a registered trademark of Oracle and/or its affiliates.

OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission by Khronos.

*Other names and brands may be claimed as the property of others.

Copyright © 2010-2015, Intel Corporation. All rights reserved.
2 Introducing the SDK

The Intel® RealSense™ SDK is a library of pattern detection and recognition algorithm implementations exposed through standardized interfaces. The library aims at lowering barriers to using these algorithms and shifting the application developers' focus from coding the algorithm details to innovating on the usage of these algorithms for next generation human computer experience.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor- dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

This document addresses application development setup and product packaging details.

Notational Conventions

This SDK document uses the Intel Clear typeface for normal prose.

With the exception of section headings, captions and the table of contents, all code-related items appear in the Courier New typeface (Status).

Hyperlinks appear underlined in blue, such as Status.

This is a note that provides additional information to aid your understanding of the procedure or concept.

This is a tip that provides alternate methods or shortcuts.

This is a result statement which indicates what you can expect to see or happen after performing a step.
3 Getting Started with the SDK

The SDK requires the following hardware and software setup:

**Hardware requirements:**
- 4th generation Intel® Core™ processors based on the Intel microarchitecture code name Haswell
- 8GB free hard disk space
- Intel® RealSense™ 3D camera (required to connect to a USB* 3 port)

**Software requirements:**
- Microsoft® Windows® 8.1 or 10 OS 64-bit
- Microsoft Visual Studio® 2010-2015 with the latest service pack
- Microsoft .NET® 4.0 Framework for C# development
- Unity® PRO 4.1.0 or higher for Unity game development
  - Starting Unity 4.6.3 patch 1, you can use the Personal version.
- Any of the following browsers for JavaScript* development:
  - Microsoft Internet Explorer® 11.0.9600
  - Microsoft Edge® 20.10240.16384.0
  - Google® Chrome® 42.0.2311
  - Mozilla® Firefox® 37.0
- Processing® 2.1.2 or higher for Processing development
- Java® JDK 1.7.0_11 or higher for Java development
- OpenCL™ 1.2 required for the Scene Perception and Enhanced Photography & Videography modules
3.1 Installing the SDK

To install the SDK software, complete the following steps:


2. You will see a welcome screen as illustrated in Figure 1. Follow the installer instructions to complete the installation process.

   By default, the SDK installs to the C:/Program Files (x86)/Intel/RSSDK directory.

   If the SDK installer detects any existing SDK versions, the SDK installer will prompt you for an upgrade. It is recommended to always do a clean uninstall and then install any newer SDK versions.

3. After installation, reboot the system when prompted. This step is critical to propagate all environmental variables.
3.2 Setting Up the Camera

To setup the camera, complete the following steps. If the camera is already integrated into the computer or laptop, skip to step 3.

1. Install the camera on top of the computer or laptop lid.
2. Plug the USB connector into one of the USB3 ports, as illustrated in Figure 1.

3. Position yourself comfortably, with your back supported by the chair in a relaxed position, so that your hands can move freely in front of the camera.

💡 To avoid fatigue, it is critical to find a relaxed position that satisfies the above requirement.

Here are the steps to check if the SDK and the camera are installed correctly:

1. From the Startup menu, select Intel® RealSense™ SDK>Tools>Camera Explorer to launch the CameraExplorer application.

2. Your screen looks similar to Figure 2. Click on the camera tab to stream the color and depth streams.

✔️ If your screen looks similar to the one shown in Figure 2, the camera is working as expected.
Figure 2: Camera Explorer Camera Selection Window

Figure 3: Camera Explorer Streaming Window
3.3 Configuring Development Environment

This section describes how to configure the development environment.
3.3.1 Configuring C++ Development Environment

The SDK provides two ways to setup the C++ development environment:

- For easy integration, the application can use the integration [Property Sheets][13].
- For additional flexibility, the application can directly set the [Project Settings][14].
3.3.1.1 Importing the SDK Property Sheets

For easy integration into the Microsoft Visual Studio development environment, use the property sheets located under the $(RSSDK_DIR)/props directory. Table 1 lists available property sheets.

<table>
<thead>
<tr>
<th>Property Sheet</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VS2010-15.Integration.MD.props</td>
<td>Microsoft Visual Studio 2010-2015 property sheet for applications that compile with the dynamic runtime option.</td>
</tr>
<tr>
<td>VS2010-15.Integration.MT.props</td>
<td>Microsoft Visual Studio 2010-2015 property sheet for applications that compile with the static runtime option.</td>
</tr>
</tbody>
</table>

Table 1: Visual Studio Integration Property Sheets

To import the property sheets, complete the following steps:

- Create a new project or open an existing project.
- Open the property manager by View → Other Windows → Property Manager.
- Right click on the project name and choose Add Existing Property Sheet. Add VS2010-15.Integration.MD.props or VS2010-15.Integration.MT.props for the application that requires dynamic or static runtime, respectively.
3.3.1.2 Configuring Project Settings

Alternatively, you can directly modify the development settings as follows:

- Include Paths:

  Add any required include paths from Table 2 to the respective project setting.

<table>
<thead>
<tr>
<th>Include Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(RSSDK_DIR)/include</td>
<td>Required to access I/O module and algorithm module functions</td>
</tr>
<tr>
<td>$(RSSDK_DIR)/sample/common/include</td>
<td>Required to access any utility class functions</td>
</tr>
</tbody>
</table>

Table 2: Include Paths

- Library Files and Library Paths:

  For applications that compile with the static runtime option, the SDK provides prebuilt libraries listed in Table 3. Add any required library paths and library files to the respective project setting.

<table>
<thead>
<tr>
<th>Library Path</th>
<th>Library File</th>
<th>Prebuilt Library Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(RSSDK_DIR)/lib/$(PlatformName)</td>
<td>libpxc.lib (RELEASE) libpxc_d.lib (DEBUG)</td>
<td>SDK library with the static runtime option</td>
</tr>
<tr>
<td>$(RSSDK_DIR)/sample/common/lib/$(PlatformName)/$(PlatformToolset)</td>
<td>libpxcutils.lib (RELEASE) libpxcutils_d.lib (DEBUG)</td>
<td>SDK utility library with the static runtime option</td>
</tr>
</tbody>
</table>

Table 3: Library Paths and Library Files

For applications that compile with the dynamic runtime option, the SDK provides prebuilt libraries listed in Table 4. Add any required library paths and library files to the respective project setting.

<table>
<thead>
<tr>
<th>Library Path</th>
<th>Library File</th>
<th>Prebuilt Library Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(RSSDK_DIR)/lib/$(PlatformName)</td>
<td>libpxcmd.lib (RELEASE) libpxcmd_d.lib (DEBUG)</td>
<td>SDK library with the dynamic runtime option</td>
</tr>
<tr>
<td>$(RSSDK_DIR)/sample/common/lib/$(PlatformName)/$(PlatformToolset)</td>
<td>libpxcutilsmd.lib (RELEASE) libpxcutilsmd_d.lib (DEBUG)</td>
<td>SDK utility library with the dynamic runtime option</td>
</tr>
</tbody>
</table>

Table 4: Library Paths and Library Files
For any special compilation options that are not covered by the prebuilt libraries, you can re-compile library source files that are listed in Table 5.

<table>
<thead>
<tr>
<th>Source Files</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(RSSDK_DIR)/src/libpxc/*.cpp</td>
<td>SDK library source files</td>
</tr>
<tr>
<td>$(RSSDK_DIR)/sample/common/src/*.cpp</td>
<td>SDK utility library source files</td>
</tr>
</tbody>
</table>

Table 5: Library and Utility Library Source Files
3.3.2 Configuring C# Development Environment

The SDK exposes the C# interfaces as two dynamically linked libraries (DLL), which support Microsoft .NET framework 4.0.

- **C# interface DLL**: libpxcclr.cs.dll
- **C++ P/Invoke DLL**: libpxccpp2c.dll

Use the following steps to add DLL as a reference in Microsoft Visual Studio 2010-2012:

- Create a new project or open an existing project.
- In the solution explorer, right click the project name and select Add Reference.
- Add $(RSSDK_DIR)/bin/win32/libpxcclr.cs.dll or $(RSSDK_DIR)/bin/x64/libpxcclr.cs.dll as the reference.

It is a known limitation that Microsoft Visual Studio cannot handle 32-bit and 64-bit references at the same time. Thus the application must explicitly modify the reference before building a different target. See this forum post for a possible workaround.

💡 If the application uses Copy Local=True, the application must manually copy libpxccpp2c.dll to your local directory.
3.3.2.1 Customize the C# libraries

You can customize the C# wrapper libraries to add any missing features, such as certain format conversion functions for C# applications.

The source code of the `libpxcclr.cs.dll` is under `$(RSSDK_DIR)/framework/common/pxcclr.cs`.

- Click `libpxcclr.cs_vs2010-15.sln` to launch the solution.
- Modify the code to add any missing features.
- Build and save the rebuilt library `libpxcclr.cs.dll`, under the local `bin` directory.

The source code of the library `libpxccpp2c.dll` is under `$(RSSDK_DIR)/framework/common/pxccpp2c`.

- Click the corresponding solution file, for example, `libpxccpp2c_vs2012.sln` to launch the solution.
- Modify the code to add any missing features.
- Build and save the library `libpxccpp2c.dll`, under the local `bin` directory.
3.3.3 Configuring Unity Development Environment

Complete the following steps to configure the environment for Unity game application development:

- Create a new Unity project.

- Import the Unity package $(RSSDK_DIR)/framework/Unity/UnityCSharp.unitypackage, or do it manually as follows:
  - Create a Plugins directory under Assets.
  - Copy the following file(s) from $(RSSDK_DIR)/bin/win32 to Plugins\x86:
    - libpxccpp2c.dll: This file is the P/Invoke library of the SDK functions.
    - libpxccpp2c.dll.signature: This is an optional signature file that enables the Unity Personal version after v4.6.3p1.
  - Copy the following file(s) from $(RSSDK_DIR)/bin/x64 to Plugins\x86_64:
    - libpxccpp2c.dll: This file is the P/Invoke library of the SDK functions.
    - libpxccpp2c.dll.signature: This is an optional signature file that enables the Unity Personal version after v4.6.3p1.
  - Copy the following file(s) from $(RSSDK_DIR)/bin/win32 or $(RSSDK_DIR)/bin/x64 and $(RSSDK_DIR)/framework/common/pxcclr.cs/src:
    - libpxcclr.unity.dll: This file is the Unity-wraper of the SDK functions.
    - pxcmddefs.extensions.cs: This optional file that provides additional type extension functions.
3.3.3.1 Customizing the Unity Libraries

You can customize the supporting libraries for Unity to add any missing features, such as certain format conversion functions for Unity applications.

The source code of the library libpxcclr.unity.dll is under $(RSSDK_DIR)/framework/common/pxcclr.cs.

- Click libpxcclr.unity_vs2010-15.sln to launch the solution.
- Modify the code to add any missing features.
- Build and save the rebuilt library libpxcclr.unity.dll, under the local bin directory.

The source code of the library libpxccpp2c.dll is under $(RSSDK_DIR)/framework/common/pxccpp2c.

- Click the corresponding solution file, for example, libpxccpp2c_vs2012.sln to launch the solution.
- Modify the code to add any missing features.
- Build and save the library libpxccpp2c.dll, under the local bin directory.

Any customized build works with the Unity Pro version (for the plugin feature.)
3.3.4 Configuring Processing Development Environment

Do the following steps to setup the environment for Processing application development:

- Copy everything under \$(RSSDK_DIR)/framework/Processing/libraries to the Processing sketch directory.

💡 In the Processing sketch, import the SDK name space by "import intel.rssdk.*;"
3.3.5 Configuring Java Development Environment

To setup the Java environment, copy all files under $(RSSDK_DIR)/framework/common/pxcclr.java/bin/* to the Java application directory.

In the Java application, import the SDK name space by "import intel.rssdk.*;"

To build and run the Java application, use the following scripts:

```
javac -classpath libpxcclr.java.jar *.java
java -classpath libpxcclr.java.jar;.<java-class>
```

where <java-class> is the main Java class of the application.
3.3.6 Configuring JavaScript Development Environment

No special setup is needed for JavaScript application development.

In your JavaScript application, reference to the following libraries:

- **Promise**: If not natively supported by your browser, you can download Promise from https://www.promisejs.org/polyfills/promise-done-6.0.0.min.js.

- **autobahn**: You can download autobahn from many browser hosting sites, for example, https://autobahn.s3.amazonaws.com/autobahnjs/latest/autobahn.min.js.

- $(RSSDK_DIR)/framework/common/JavaScript/realsense.js

See Also

[Deploying JavaScript Applications](#)
4 Deploying Your Application

Use the following flow diagram to prepare your application for deployment:

For C++ applications,
- Install the SDK runtimes. See Installing SDK runtimes.

For C#, Java, Processing, and Unity C# applications,
- Package the language wrappers. See Packaging Supporting Libraries.
- Install the SDK runtimes. See Installing SDK runtimes.

For JavaScript applications
- Prepare the landing page and WebApp runtime. See Deploying JavaScript Applications.

See Also
Related Topics
Privacy Notice Guidelines
4.1 Packaging Supporting Libraries

The SDK provides the following framework supporting libraries. You should package them within your application:

C++

N/A

C# .NET4

$(RSSDK_DIR)/bin/win32:
libpxcclr.cs.dll
libpxcpp2c.dll

$(RSSDK_DIR)/bin/x64:
libpxcclr.cs.dll
libpxcpp2c.dll

Unity C#

$(RSSDK_DIR)/bin/win32:
libpxcclr.unity.dll
libpxccpp2c.dll
libpxccpp2c.dll.signature

$(RSSDK_DIR)/bin/x64:
libpxcclr.unity.dll
libpxccpp2c.dll
libpxccpp2c.dll.signature

Java

$(RSSDK_DIR)/framework/common/pxcclr.java/bin:
libpxcclr.java.jar
libpxcclr.jni32.dll
libpxcclr.jni64.dll

Processing

$(RSSDK_DIR)/framework/common/pxcclr.java/bin:
libpxcclr.processing.jar
libpxcclr.jni32.dll
libpxcclr.jni64.dll

JavaScript

$(RSSDK_DIR)/framework/common/JavaScript:
realsense.js

C# UAP

N/A
4.2 Installing SDK runtimes

You need to install the SDK runtimes during your application deployment. Table 6 lists available SDK runtime installers.

<table>
<thead>
<tr>
<th>Runtime Installer (YYYY is the SDK version)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>intel_rs_sdk_runtime_YYYY.exe</td>
<td>The SDK runtime master installer for offline installation.</td>
</tr>
<tr>
<td></td>
<td>- This installer is available through download due to its size.</td>
</tr>
<tr>
<td>intel_rs_sdk_runtime_websetup_YYYY.exe</td>
<td>The SDK runtime master installer for online installation.</td>
</tr>
<tr>
<td>intel_rs_sdk_runtime_core_YYYY.exe</td>
<td>The SDK core runtime installer.</td>
</tr>
<tr>
<td>intel_rs_sdk_runtime_webapp_YYYY.exe</td>
<td>The SDK WebApp runtime installer specific for the JavaScript applications. See Deploying JavaScript Applications.</td>
</tr>
</tbody>
</table>

Table 6: The SDK Runtime List

You can pack the runtime installer(s) in your application installer using any of the following installation scenarios:

Installation Case 1: Capture Only Installation

The SDK core runtime installer includes libraries for raw camera data capturing and SDK essential features such as session management, SenseManager pipelining, and file recording and playback. If your application works with camera data capturing only and does not use any algorithms (such as face tracking or hand tracking), you can pack the core runtime installer as part of the application installation.

Installation Case 2: Module Offline Installation

The SDK runtime master installer includes the core runtime and additionally all algorithm (module) runtimes for offline installation. Usually an application only uses a subset of the provided algorithms. The master installer can export a set of customized installation scripts that include only the algorithms needed.

To generate the customized installation scripts, perform the following steps:

- Run the SDK runtime master installer with the command line `--pre-bundle=<dir>`, where `<dir>` is the absolute path of where the installer script should be exported.
- Go through the installation flow and select the modules that your application needs. No
installation is done. Instead, the master installer exports the installation scripts to the
destination directory.

- Pack the entire destination directory into your application installer, and follow the
  commands in readme_cmd.txt to install the SDK runtimes.

Installation Case 3: Module Online Installation

To minimize the size of the application, you can use the SDK runtime master installer for
online installation. The installer downloads the required modules from the Intel website
during installation. Use the --finstall=<feature-list> --fnone=all command line to
customize the algorithms to be installed. See Features and Components[5] for the feature
list.

Important Notes

- The runtime installer may have different distribution rights. See redist.txt in the SDK
  package for details.

- The runtime installer requires elevated privilege to install the runtime libraries and data
  files into the system folders.

- The installation process may take some time. Your application installer must wait for the
  installation process to complete.

- The runtime installer by default (without any command line options) goes through a set
  of installation dialogs, including asking the user to accept the Intel EULA. If this process
  must be customized for seamless installation, use the command line option --silent
  --no-progress --acceptlicense=yes. You must integrate Intel EULA terms into your
  application's EULA to ensure that the user is aware of, and agrees to, the terms. See the
  license text of the SDK license agreement for details.
4.2.1 Speech Runtime and Language Packs

The speech runtime and language packs do not have an offline installation option. They are available only as online download if not already on the system.

To add the speech runtime and the language pack(s), append the speech runtime and language pack feature names to the installer option `--finstall`. See Installer Feature Code for the feature names. For example, use `--finstall=voice,nuance_en_us_cnc` to download and install the speech runtime and the US English language pack for command and control.

If you use the Module Offline Installation installation option, select the speech components and follow the instruction in readme_cmd.txt. For example, if you select the hand tracking module, the speech runtime and the US English language pack for command and control, the instruction in readme_cmd.txt is similar to `--finstall=core,hand,voice,nuance_en_us_cnc`. The generated customized installer contains the local hand tracking runtime and the code to trigger downloading of the speech runtime and the language pack.

During the runtime installation, due to license restriction, the installer needs to validate the Intel® RealSense™ 3D Camera platform. You need to prompt the user to plug in the camera if the camera is not integrated. The installer returns status code 3020 (see Command Option and Status Code) and skips the speech component installation if the installer fails to validate the platform.
4.2.2 Uninstalling SDK runtimes

The SDK runtimes are shared among multiple applications. The application installer should never remove any SDK runtimes. Leave them on the system.

The user can manually uninstall the SDK runtime from Control Panel → Programs and Features. Do so only after all SDK applications are uninstalled from the system.
4.2.3 Installer Options

Installer Command Option

You can use the following command line options with the SDK runtime installers:

--silent --no-progress --acceptlicense=yes  
Run the installer in the silent mode.

You must integrate Intel EULA terms into your application's EULA. See the license text of the SDK license agreement for details.

--pre-bundle=<dir>  
Create an installer under <dir> that installs the selected SDK features. <dir> must be an absolute path.

--finstall=<feature-list> --fnone=all  
Install only the requested features. Use comma to separate multiple features. See Features and Components for the feature definitions.

--finstall=all  
Install all runtimes.

--extract-to=<dir>  
Unpack to the specified directory. The default directory is %TEMP%\{GUID}.

--f200 --sr300 --f200 --r200 --front --rear  
Force a camera model check such that the runtime is installed only when the requested camera model is detected. If the camera model is not detected, the installer aborts with status code 1633.

The --front option checks for any front facing camera and --rear option for any rear facing camera.

The option --finstall=all or --finstall=<feature-list> must present in the command options for the installer to install the runtime(s).

Examples

Install the core runtime silently  
intel_rs_sdk_runtime_core_YYYY.exe --silent --no-progress --acceptlicense=yes --finstall=all

Install the face runtime silently via web download  
intel_rs_sdk_runtime_websetup_YYYY.exe --silent --no-progress --acceptlicense=yes --finstall=core,face3d --fnone=all

See Also
Installer Feature Code

Installer Status Code
4.2.4 Installer Feature Code

To describe different installation items, the SDK runtime installer uses the concept Feature to describe the set of installations that enable certain SDK functionality. For example, the core feature installs the runtime libraries needed for color and depth data capturing. Each feature includes the 32-bit and 64-bit runtime dynamic libraries and the data files shared between 32-bit and 64-bit applications.

Use Feature in the following case(s):

- **In the application installer:** For example, specify `--finstall=hand` to install the hand tracking module: its runtime libraries and data files.

<table>
<thead>
<tr>
<th>Camera</th>
<th>Feature Name</th>
<th>Feature Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Both</td>
<td>core</td>
<td>The SDK essentials including camera data capturing. This is must have for any SDK applications.</td>
</tr>
<tr>
<td>F200</td>
<td>blob</td>
<td>The blob/contour extraction algorithm.</td>
</tr>
<tr>
<td>F200</td>
<td>emotion</td>
<td>The emotion detection algorithm and data files.</td>
</tr>
<tr>
<td>R200</td>
<td>epv</td>
<td>The enhanced photography and videography algorithms.</td>
</tr>
<tr>
<td>Both</td>
<td>face3d</td>
<td>The face tracking algorithm and data files.</td>
</tr>
<tr>
<td>F200</td>
<td>hand</td>
<td>The hand tracking algorithm and data files.</td>
</tr>
<tr>
<td>F200</td>
<td>personify</td>
<td>The user segmentation algorithm.</td>
</tr>
<tr>
<td>R200</td>
<td>scene_perception</td>
<td>The scene perception algorithm and data files.</td>
</tr>
<tr>
<td>Both</td>
<td>scan3d</td>
<td>The 3D scanning algorithm and data files.</td>
</tr>
<tr>
<td>F200</td>
<td>touchlesscontroller</td>
<td>The touchless controller algorithm and data files.</td>
</tr>
<tr>
<td>F200</td>
<td>track3d_metaio</td>
<td>object tracking algorithm.</td>
</tr>
<tr>
<td>F200</td>
<td>utils</td>
<td>The SDK utility functions such as data smoothing.</td>
</tr>
<tr>
<td>Both</td>
<td>voice</td>
<td>The speech recognition and synthesis algorithms.</td>
</tr>
</tbody>
</table>
The speech feature requires at least one speech language pack to be installed on the system.

<table>
<thead>
<tr>
<th>Language Pack</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nuance_en_us_cnc nuance_en_us_dict nuance_en_us_vocal</td>
<td>The US English language pack for command and control (cnc), dictation (dict), and text to speech (vocal), respectively.</td>
</tr>
<tr>
<td>nuance_en_gb_cnc nuance_en_gb_dict nuance_en_gb_vocal</td>
<td>The British English language pack for command and control (cnc), dictation (dict), and text to speech (vocal), respectively.</td>
</tr>
<tr>
<td>nuance_es_la_cnc nuance_es_la_dict nuance_es_la_vocal</td>
<td>The Latin American Spanish language pack for command and control (cnc), dictation (dict), and text to speech (vocal), respectively.</td>
</tr>
<tr>
<td>nuance_pt_br_cnc nuance_pt_br_dict nuance_pt_br_vocal</td>
<td>The Portuguese language pack for command and control (cnc), dictation (dict), and text to speech (vocal), respectively.</td>
</tr>
<tr>
<td>nuance_de_de_cnc nuance_de_de_dict nuance_de_de_vocal</td>
<td>The German language pack for command and control (cnc), dictation (dict), and text to speech (vocal), respectively.</td>
</tr>
<tr>
<td>nuance_fr_fr_cnc nuance_fr_fr_dict nuance_fr_fr_vocal</td>
<td>The French language pack for command and control (cnc), dictation (dict), and text to speech (vocal), respectively.</td>
</tr>
<tr>
<td>nuance_it_it_cnc nuance_it_it_dict nuance_it_it_vocal</td>
<td>The Italian language pack for command and control (cnc), dictation (dict), and text to speech (vocal), respectively.</td>
</tr>
<tr>
<td>nuance_ja_jp_cnc nuance_ja_jp_dict nuance_ja_jp_vocal</td>
<td>The Japanese language pack for command and control (cnc), dictation (dict), and text to speech (vocal), respectively.</td>
</tr>
<tr>
<td>nuance_zh_cn_cnc nuance_zh_cn_dict nuance_zh_cn_vocal</td>
<td>The Chinese language pack for command and control (cnc), dictation (dict), and text to speech (vocal), respectively.</td>
</tr>
</tbody>
</table>
4.2.5 Installer Status Code

The SDK runtime installers return the following status codes to the application installer:

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The installation was completed successfully.</td>
</tr>
<tr>
<td>1</td>
<td>The installation was aborted.</td>
</tr>
<tr>
<td>2</td>
<td>The installation was canceled by the user.</td>
</tr>
<tr>
<td>3</td>
<td>The installation was completed with some errors in the optional component installation.</td>
</tr>
<tr>
<td>1633</td>
<td>The installation was aborted due to unsupported platform, OS, or camera model.</td>
</tr>
<tr>
<td>3010</td>
<td>The installation was completed and reboot was required. Reserved. Usually the SDK runtime does not require reboot.</td>
</tr>
<tr>
<td>3015</td>
<td>The installation was completed with the warning that the Intel® RealSense™ Depth Camera Manager installed on the system was incompatible with the SDK runtime version. The user must upgrade the Intel RealSense Depth Camera Manager for the SDK runtime to function properly.</td>
</tr>
<tr>
<td>3020</td>
<td>The installer failed to identify the platform as a valid Intel RealSense 3D Camera platform. Premium components (that had license restrictions) were not installed. Other component installation was completed successfully.</td>
</tr>
</tbody>
</table>

💡 See the installer log under `%TEMP%/micl_tmp_<username>` for the detailed error description.
4.3 Deploying JavaScript Applications

Do the following steps to prepare your SDK JavaScript application for deployment:

- [Designing the Landing Page](#)
- [Preparing WebApp Runtime](#)
- [Hosting Your Application](#)
4.3.1 Designing the Landing Page

You need to design a landing page to address the following issues:

1. Check if the platform or the browser your application runs upon is prepared to run your application.

   If the platform detection shows that your platform is ready, proceed to your application context. Otherwise, to prepare the platform, display the WebApp runtime URL and instruct the end user to download and execute the WebApp runtime.

  💡 You only need to prepare the platform once. The platform detection will pass the second time the user visits your application page.

   See Platform Detection\(^{36}\) for how to detect the platform features and Preparing the WebApp Runtime\(^{37}\) for how to create the WebApp runtime specifically for your application needs.

2. Show privacy notice to the end users to comply with the SDK privacy guidelines.

   See Privacy Notice Guidelines\(^{34}\).

Platform Detection

The followings are prerequisites of running your SDK JavaScript application:

i. The platform and OS must be supported.

ii. The browser must support JavaScript and Web Socket\(^{*}\).

iii. The platform must be equipped with the model of the Intel® RealSense™ 3D Camera that your application is designed for.

iv. The right version of the camera driver is installed.

v. The right version of the SDK runtime(s) is installed.

Use the `detectPlatform` function of the `SenseManager` interface to check the platform features. The function takes two arguments:

1. A list of SDK runtimes that your application must use. For example, if you use the SDK face tracking feature, specify the feature name: "face3d". See Installer Feature Code\(^{31}\) for the full list of SDK runtimes.

   💡 Due to privacy concerns, the SDK only exposes those runtime features that do not involve any personally identifiable information: blob tracking, face tracking, hand tracking and speech command and control. Image or audio accesses are explicitly disabled.

2. The optional list of camera models that your application must work with. For example,
specify "f200" to specify that your application can work with the camera model F200. You can also specify the camera family such as "front" and "rear". See Camera Check.

If ignored, the function checks if any front or rear camera exists.

The detectPlatform function returns the detection results as a promise object. The nextStep parameter indicates what the application must do next. The value is as follows:

- "driver": The camera driver is not installed or out dated. Instruct the user to install/uprade the camera driver.
- "runtime": The SDK runtime is not installed or obsolete. Display the WebApp runtime URL and instruct the user to download and install the SDK runtime(s).
- "unsupported": The platform, OS, or browser is not supported.
- "ready": Everything is ready to run your application. Proceed to your application context.

Example 1 shows a typical platform checking routine.

Example 1: The Application Landing Page Script: Platform Detection

```javascript
// name space shortcut
var rs=intel.realsense;

// detect platform: checking for blob and front camera F200
rsSenseManager.detectPlatform(['blob'], ['f200']).then(function (flags) {
  if (flags.nextStep == 'driver') {
    document.write("Please update your camera driver from your computer manufacturer.");
    return;
  }
  if (flags.nextStep == 'unsupported') {
    document.write("The platform is not supported.");
    return;
  }
  if (flags.nextStep == 'runtime') {
    document.write("Download & install SDK runtime(s): URL");
    return;
  }
  if (flags.nextStep == 'ready') {
    // The platform is ready. Proceed to the application context.
    return;
  }
}).catch(function (e) {
  document.write("Detection Failure: " + e);
});
```
4.3.2 Preparing the WebApp Runtime

The SDK WebApp runtime is a small executable (<2MB) that performs the following operations:

- Check if the SDK runtimes are installed on the system. If not, download the SDK runtimes and install them.

💡 To improve user experience, the SDK WebApp runtime will not re-download any runtimes that are already installed on the system.

- Optionally check if the camera meets the application needs.

The SDK WebApp runtime is located at $(RSSDK_DIR)/runtime/intel_rs_sdk_runtime_webapp_<version>.exe. You can use the WebApp runtime in any of the following ways:

**Using the WebApp Runtime As is**

By default, the SDK WebApp runtime installs the following SDK runtimes and language model:

- Blob Tracking
- Face Tracking
- Hand Tracking
- Speech Command Control
- The English speech command and control language model

**Customizing the WebApp Runtime with Installation Options**

If your application does not use all features, you can customize the WebApp runtime installation options to reduce the time it takes to prepare the platform, thus improving user experience. Do the following to customize the SDK WebApp runtime:

1. Launch the HTML utility $(RSSDK_DIR)/framework/common/JavaScript/webapp_cust.html. Select the features for your application. The utility encodes your choice into the runtime filename and display it.

2. Copy the SDK WebApp runtime executable $(RSSDK_DIR)/runtime/intel_rs_sdk_runtime_webapp_<version>.exe to any writable location. Rename the file as displayed in step 1.
The SDK WebApp Runtime Customizer

Select the modules you wish to use in your JavaScript® app and click the "Generate Installer Name" button.

Copy the SDK webapp runtime installer located at $(RSSDK_DIR)/runtime to a writable location and rename it to this name. The customized installer only installs the modules that you selected.

Modules required for app:

- The Face Tracking Module
- The Hand Tracking Module
- The Blob Tracking Module
- Speech Command & Control - English
- Speech Command & Control - German
- Speech Command & Control - Spanish
- Speech Command & Control - French
- Speech Command & Control - Italian
- Speech Command & Control - Japanese
- Speech Command & Control - Portuguese
- Speech Command & Control - Chinese (Mandarin)

Cameras required:

- Intel® RealSense™ 3D Camera, model F200
- Intel® RealSense™ 3D Camera, model R200
- Intel® RealSense™ 3D Camera, model SR300
- Any front-facing camera
- Any rear-facing camera

Generate Installer Name
rs_sdk_2532508_webapp_v6.exe

For example, as illustrated in Figure 4, if you use blob tracking only in your application and want to additionally check camera model F200, rename the file to be

rs_sdk_2532508_webapp_v6.exe

The sub string ".2532508." enables the WebApp runtime customization. The rest file name string is not critical to the customization.

You can now use the renamed WebApp runtime for your application.

Reworking the WebApp Runtime GUI (Advanced)

If the WebApp runtime GUI style does not fit with your application, you can run the WebApp runtime in the silent mode and overlay your own installer GUI on top. For example, a puppy game may want to add some pinky taste to the installer screens.

See Installing SDK runtimes for the general concept, Installer Options for how to specify the feature options, and Installer Status Code for any error handling.
4.3.3 Hosting Your Application

Host the following files onto your website:

- $(RSSDK_DIR)/framework/common/JavaScript/realsense.js
- The web runtime as described in Preparing the WebApp Runtime.
- Your application pages, style sheets, and scripts.

Your application is ready to launch.
4.4 Related Topics

This section discusses a few application deployment topics.
4.4.1 SDK Runtime Upgrade Policy

The SDK runtime contains the module (algorithm) implementations and their data files. The SDK includes runtime installers under $(RSSDK_DIR)/runtime. The directory exists if you select redistributable components, selected by default, during the SDK developer package installation.

The runtime installers perform the following operations:

- If there is no existing runtime, install the SDK modules.

  Certain SDK module runtimes may have license restrictions. See Speech Runtime and Language Packs[^1] for details.

- If there is an existing runtime, update the runtime.

  The upgrading policy is illustrated in Figure 5. If the existing runtime is older (the major version is the same and the minor version is lower), the existing runtime is updated. Otherwise the two runtimes coexist on the same system. The runtime installers do not override SDK runtimes that are up-to-date, or of a different major version.

![Figure 5: Runtime Upgrading Policies](image)

[^1]: Speech Runtime and Language Packs
4.4.2 Checking SDK, Camera Driver, and Algorithm Versions

Your application may be installed on a system that is different from your development environment. Handle the following cases gracefully in your application:

- **Camera Driver Version**: You must check the camera driver version installed on the system. The camera driver is pre-installed on any OEM systems and is not part of the SDK runtime. If the camera driver version is older, certain feature may not be available. You must adapt your application to use fewer features, (or prompt the user to upgrade as a last resort.)

💡 Check the camera driver features in the camera driver release notes.

- **Module/Algorithm Version and Availability**: Always install the SDK runtime(s), which ensures that the correct version of the SDK runtime(s) is installed on the system. Certain runtime may have license restrictions to install only on certain type of systems. It is recommended that you also check the module availability at run time.

💡 The SDK runtime installers skip installation if the system already has the up-to-date runtimes. No need to redo the check in your application installer.

**Checking Versions during Installation**

You can retrieve the camera driver version from the registry keys in the following table:

<table>
<thead>
<tr>
<th>Camera Model</th>
<th>Registry Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>F200</td>
<td>HKLM/Software/Intel/RSSDK/Components/ivcam/Version</td>
</tr>
<tr>
<td>R200</td>
<td>HKLM/Software/Intel/RSSDK/Components/dcm_r200/Version</td>
</tr>
</tbody>
</table>

The camera driver version is stored as a string in the form of major.minor.build.revision. Usually it is sufficient to check only the major and minor numbers.

You can check whether an algorithm module is installed and retrieve its version from the registry key: HLKM/Software/Intel/RSSDK/v<major-version>/Components/<feature-code>/Version, where major-version is the SDK interface major version number, and feature-code denotes each algorithm feature. See Installer Feature Code for a list.

For example, the blob module version can be retrieved by looking at HLKM/Software/Intel/RSSDK/v5/Components/blob/Version, as illustrated in Figure 6. The version string is in the form of major.minor.build.revision.
Checking Versions at Run Time

After the SDK runtime(s) is installed, you can check the camera driver version and algorithm/module versions at run time.

Example 2 shows how to check the SDK interface version.

Example 2: Get SDK Interface Version

\[\text{C++}\]

```cpp
// session is a PXCSession instance
PXCSession::ImplVersion sdk_version=session->QueryVersion();
```

\[\text{C#}\]

```csharp
// session is a PXCMSession instance
PXCMSession.ImplVersion sdk_version=session.QueryImplVersion();
```

\[\text{Java}\]

```java
// session is a PXCMSession instance
PXCMSession.ImplVersion sdk_version=session.QueryImplVersion();
```

\[\text{JavaScript}\]

```javascript
// sm is a SenseManager instance
var version=sm.session.version;
```

You can retrieve the camera driver version and any algorithm module version from the corresponding module descriptor. In the case of the camera driver, use the capture module descriptor.

Assuming the camera and the algorithm module(s) are properly configured in the SenseManager pipeline, Example 3 shows how to check the camera driver version and the face module version. In all other cases, the EnableFace function or the Init function will return an error code, which indicates that the face module or the camera is not available.

Example 3: Check Camera Driver Version

\[\text{C++}\]

```cpp
// session is a PXCSession instance
PXCSession::ImplVersion sdk_version=session->QueryVersion();
```
PXCSession::ImplVersion GetVersion(PXCSession *session, PXCBase *module) {
    PXCSession::ImplDesc mdesc={};
    session->QueryModuleDesc(module, &mdesc);
    return mdesc.version;
}

// sm is the PXCSenseManager instance
PXCSession::ImplVersion driver_version=GetVersion(sm->QuerySession(), sm->QueryCaptureManager());
PXCSession::ImplVersion face_version=GetVersion(sm->QuerySession(), sm->QueryFace());

(C#)

PXCMSession.ImplVersion GetVersion(PXCMSession session, PXCMBase module) {
    PXCMSession.ImplDesc mdesc;
    session.QueryModuleDesc(module, out mdesc);
    return mdesc.version;
}

// sm is the PXCSenseManager instance
PXCMSession.ImplVersion driver_version=GetVersion(sm.session, sm.captureManager.capture);
PXCMSession.ImplVersion face_version=GetVersion(sm.session, sm.QueryFace());

(Java)

PXCMSession.ImplVersion GetVersion(PXCMSession session, PXCMBase module) {
    PXCMSession.ImplDesc mdesc=new PXCMSession.ImplDesc();
    session.QueryModuleDesc(module, mdesc);
    return mdesc.version;
}

// sm is the PXCSenseManager instance
PXCMSession.ImplVersion driver_version=GetVersion(sm.QuerySession(), sm.QueryCaptureManager().QueryCapture());
PXCMSession.ImplVersion face_version=GetVersion(sm.QuerySession(), sm.QueryFace());

(JavaScript)

function GetVersion(session, module) {
    return session.queryModuleDesc(module).then(function(mdesc) {
        return mdesc.version;
    })
}

// sm is a SenseManager instance
var driver_version;
GetVersion(sm.session, sm.captureManager.capture).then(function(version) {
    driver_version=version;
});

// face is a FaceModule instance (from FaceModule.createInstance)
var face_version;
GetVersion(sm.session, face).then(function(version) {
    face_version=version;
});
4.4.3 Algorithm Operating Ranges

The following table summarizes the operating range of the SDK algorithms:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Camera F200 Detection (cm)</th>
<th>Camera R200 Detection (cm)</th>
<th>Camera F200 Tracking (cm)</th>
<th>Camera R200 Tracking (cm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Face: Detection</td>
<td>25–75</td>
<td>50–250</td>
<td>30–100</td>
<td>55–250</td>
</tr>
<tr>
<td>Face: Landmark</td>
<td>30–100</td>
<td>50–150</td>
<td>30–100</td>
<td>50–150</td>
</tr>
<tr>
<td>Face: Expression</td>
<td>30–100</td>
<td>30–100</td>
<td>30–100</td>
<td>30–100</td>
</tr>
<tr>
<td>Face: Pulse</td>
<td>30–60</td>
<td>30–60</td>
<td>30–70</td>
<td>30–70</td>
</tr>
<tr>
<td>Face: Pose</td>
<td>30–100</td>
<td>30–100</td>
<td>50–150</td>
<td>50–150</td>
</tr>
<tr>
<td>Blob Segmentation</td>
<td>20–85</td>
<td>50–300</td>
<td>20–85</td>
<td>50–300</td>
</tr>
<tr>
<td>Hand: Segmentation** &amp;&lt;sup&gt;D&lt;/sup&gt;</td>
<td>20–80</td>
<td>NA</td>
<td>20–80</td>
<td>NA</td>
</tr>
<tr>
<td>Hand: Tracking** &amp;&lt;sup&gt;B&lt;/sup&gt;</td>
<td>20–60</td>
<td>NA</td>
<td>20–60</td>
<td>NA</td>
</tr>
<tr>
<td>Hand: Gesture** &amp;&lt;sup&gt;B&lt;/sup&gt;</td>
<td>20–60</td>
<td>NA</td>
<td>20–60</td>
<td>NA</td>
</tr>
<tr>
<td>Object Tracking</td>
<td>30–180* &amp;&lt;sup&gt;D&lt;/sup&gt;</td>
<td>NA</td>
<td>30–180*</td>
<td>NA</td>
</tr>
<tr>
<td>Emotion</td>
<td>25–100</td>
<td>NA</td>
<td>25–100</td>
<td>NA</td>
</tr>
</tbody>
</table>

* Require at least 5000 pixels at 640x480 for proper detection, which translates to

** Require a minimum palm size of 5.5 cm (about a fix-year old kid palm size) and hand interaction speed within the following range:

<table>
<thead>
<tr>
<th>Distance</th>
<th>Minimum Rectangular Object Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>30 cm</td>
<td>4.5 cm x 3.5 cm</td>
</tr>
<tr>
<td>60 cm</td>
<td>9 cm x 7 cm</td>
</tr>
<tr>
<td>100 cm</td>
<td>16 cm x 11 cm</td>
</tr>
<tr>
<td>150 cm</td>
<td>23 cm x 18 cm</td>
</tr>
<tr>
<td>180 cm</td>
<td>28 cm x 21 cm</td>
</tr>
</tbody>
</table>

** Requirement for proper object detection and tracking speed.
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>HVGA</td>
<td>2m/s</td>
</tr>
<tr>
<td>VGA</td>
<td>0.75m/s</td>
</tr>
</tbody>
</table>
4.4.4 Working with Multiple Modalities

The SDK includes multiple algorithms (modalities or modules, used interchangeably) and each extends your application's interaction with the user. When enabling multiple modalities in your application, you must observe the following limitations/considerations:

Stream Resolution and Frame Rate

Multiple modalities must agree on the stream resolutions and frame rates. For example, a face tracking module that needs color 1920x1080x30fps cannot work together with the object tracking module that works only with color 640x480x30fps. An agreed upon stream configuration must be set for the two modules to function together. The configuration is stream based thus a module that uses the color stream does not conflict with a module that works with a depth stream.

The following table shows the modules and their available configurations:

<table>
<thead>
<tr>
<th>Camera F200</th>
<th>Color Resolution</th>
<th>Depth Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>width</td>
<td>320  640  640  960  1280  1920</td>
<td>320  640  640</td>
</tr>
<tr>
<td>height</td>
<td>240  360  480  540  720  1080</td>
<td>240  480  240</td>
</tr>
</tbody>
</table>

| Face Tracking (2D) | X | NA | NA | NA |
| Face Tracking (3D) | X | X | X | X | X | X |
| Hand Tracking     | NA | NA | NA | NA | NA | X | X | X |
| User segmentation | X | X | X | X | X | X |
| Emotion Detection | X | X | X | NA | NA | NA |
| Object Tracking (2D) | X | X | X | X | X | NA | NA | NA |
| Object Tracking (3D) | X | X | X | X | X | X |
| Touchless Controller | NA | NA | NA | NA | NA | NA | X | X | X |
| 3D Scanning       | NA | NA | NA | NA | NA | NA | X | X | X |

<table>
<thead>
<tr>
<th>Camera R200</th>
<th>Color Resolution</th>
<th>Depth Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>width</td>
<td>320  640  1280  1920</td>
<td>320  480  628</td>
</tr>
<tr>
<td>height</td>
<td>240  480  720  1080</td>
<td>240  360  468</td>
</tr>
</tbody>
</table>

| Face Tracking | X | X | X |
| Enhanced Photography | X | X | X | X |
| Scene Perception | X | X | X | X | X | X |
This table is for reference only and may change over SDK releases. You should not hard code the settings. Instead, use SenseManager to auto-negotiate a working configuration.

The SenseManager interface auto-negotiates the stream configuration when you enable multiple modalities. The Init function returns successfully if there is an agreed upon stream configuration. Example 4 shows how to enable face and hand modules and auto-negotiate the configuration.

Example 4: Enable Two Modalities and Auto-Negotiate Configurations

(C++)

```c++
// Create a SenseManager instance.
PXCSenseManager *sm=PXCSenseManager::CreateInstance();

// Enable face & hand tracking.
sm->EnableFace();
sm->EnableHand();

// additional face and hand configuration.
...

// Init
pxcStatus sts=sm->Init();
if (sts>=PXC_STATUS_NO_ERROR) {
    // two modalities can work together.
} else {
    // conflict in modalities.
}
```

(C#)

```csharp
// Create a SenseManager instance.
PXCMSenseManager sm=PXCMSenseManager.CreateInstance();

// Enable face & hand tracking.
sm.EnableFace();
sm.EnableHand();

// additional face and hand configuration.
...

// Init
pxcmStatus sts=sm.Init();
if (sts.IsSuccessful()) {
    // two modalities can work together.
} else {
    // conflict in modalities.
}
```

(Java)

```java
// Create a SenseManager instance.
PXCMSenseManager sm=PXCMSenseManager.CreateInstance();

// Enable face & hand tracking.
sm.enableFace();
sm.enableHand();

// additional face and hand configuration.
...

// Init
pxcmsStatus sts=sm.init();
if (sts.isSuccessful()) {
    // two modalities can work together.
} else {
    // conflict in modalities.
}
```
// Create a SenseManager instance.
FXCMSenseManager sm=FXCMSenseManager.CreateInstance();

// Enable face & hand tracking.
sm.EnableFace();
sm.EnableHand();

// additional face and hand configuration.
...

// Init
pxcmStatus sts=sm.Init();
if (sts.isSuccessful()) {
    // two modalities can work together.
} else {
    // conflict in modalities.
}

(JavaScript)

// name space short cuts
var rs=intel.realsense;
var rsf=intel.realsense.face;
var rsb=intel.realsense.blob;

// Create a SenseManager instance
var sm;
rs.SenseManager.createInstance().then(function(instance) {
    sm=instance;

    // Create a face instance
    return rsf.FaceModule.createInstance(sm).then(function(fm)) {
        // Set the handler for face data
        fm.onFrameProcessed=onFaceData;

        // face configuration: omitted here for simplicity.
        ....

        // Create a blob instance
        return rsb.BlobModule.createInstance(sm).then(function(bm)) {
            // Set the handler for blob data
            bm.onFrameProcessed=onBlobData;

            // blob configuration: omitted here for simplicity.
            ....

            // Initialize SenseManager
            return sm.Init().then(function(result) {
                if (result.sts>=rs.Status.STATUS_NO_ERROR) {
                    ....
                } else {
                    .... // conflict in configurations.
                }
            });
        });
    });
});

If you use the SenseManager polling mode with two or more modalities enabled, use
AcquireFrame(false) in your main streaming loop to avoid modality interference with each other. The argument false requests that the function returns immediately when any of the modalities produces data. (A true argument forces to wait until all modalities have processed the data.) Example 5 shows how to retrieve data from multiple modalities.

**Example 5: Retrieve Data From Multiple Modalities.**

### C++

```cpp
// Create a SenseManager instance.
PXCSenseManager *sm=PXCSenseManager::CreateInstance();

// Enable face & hand tracking.
sm->EnableFace();
sm->EnableHand();

// additional face and hand configuration.
...

// Init
pxcStatus sts=sm->Init();

// The streaming loop
for (;;) {
    sts=sm->AcquireFrame(false);
    if (sts<PXC_STATUS_NO_ERROR) break;

    // check if face data is available
    PXCFaceModule *face=sm->QueryFace();
    if (face) {
        // working with face data
    }

    // check if hand data is available
    PXCHandModule *hand=sm->QueryHand();
    if (hand) {
        // working with hand data
    }

    // resume processing
    sm->ReleaseFrame();
}

// Clean up
sm->Release();
```

### C#
// Create a SenseManager instance.
PXCM SenseManager sm = PXCM SenseManager.CreateInstance();

// Enable face & hand tracking.
sm.EnableFace();
sm.EnableHand();

// additional face and hand configuration.
...

// Init
pxcmStatus sts = sm.Init();

// The main streaming loop
for (;;)
{
    sts = sm.AcquireFrame(false);
    if (sts.IsError()) break;

    // check if face data is available
    PXCM FaceModule face = sm.QueryFace();
    if (face != null) {
        // working with face data
    }

    // check if hand data is available
    PXCM HandModule hand = sm.QueryHand();
    if (hand != null) {
        // working with hand data
    }

    // resume processing
    sm.ReleaseFrame();
}

// clean up
sm.Dispose();

(Java)
// Create a SenseManager instance.
PXCMSSenseManager sm=PXCMSSenseManager.CreateInstance();

// Enable face & hand tracking.
sm.EnableFace();
sm.EnableHand();

// additional face and hand configuration.
...

// Init
pxcmStatus sts=sm.Init();

// The main streaming loop
for (;;) {
    sts=sm.AcquireFrame(false);
    if (sts.isError()) break;

    // check if face data is available
    PXCMFaceModule face=sm.QueryFace();
    if (face!=null) {
        // working with face data
    }

    // check if hand data is available
    PXCMHandModule hand=sm.QueryHand();
    if (hand!=null) {
        // working with hand data
    }

    // resume processing
    sm.ReleaseFrame();
}

// clean up
sm.close();

(JavaScript)
Device Properties

The camera device exposes many device properties to adjust the behavior of the capturing process. These device properties affect the captured image qualities. Different SDK modalities work best under different sets of device properties, due to the nature of the algorithms. For example, using a deeper smoothing setting smoothies the captured images thus is good for modalities that needs smoother edges and clean images. Since the local details are lost, such setting is not good for modalities that need to detect and respond to fast local movements.

For an application that works with a single modality, with a bit of trail and error, it is possible to find the setting that works better than others. The following table shows the depth settings for different modalities:

<table>
<thead>
<tr>
<th>Camera F200</th>
<th>Confidence Threshold</th>
<th>Filter Option</th>
<th>Laser Power</th>
<th>Motion Range Trade off</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Factory Default</td>
<td>6</td>
<td>5</td>
<td>16</td>
<td>0</td>
<td>Median</td>
</tr>
</tbody>
</table>
The table is shown only for reference. The settings may change (or relaxed) over different SDK releases as the modality algorithms improve.

When enabling multiple modalities, you need to carefully trade off what is most important, as it is often the case that the best setting of the modalities differs. You can set the device properties in the following ways:

- **Happy Median**: Choose the setting that may not be the best for each modality but good enough for the application context. This works when there is no dominant interaction. Each interaction must be accurate (unambiguous).

- **Dominant Instance**: Choose the setting that works best for the dominant interaction. For example, if the application context is to use face tracking to identify a region of interest on the screen and then use hand tracking to manipulate the objects in the region, it is possible to set the setting best for face tracking initially and switch to the setting best for hand tracking when needed.

See also [Sharing Device Properties](#) for more details how to manage device properties.

### Power and Performance

Consider power and performance as well. Each SDK modality incurs nontrivial workloads, although the SDK continues to optimize for the platform. No data to share as this is highly platform and application context dependent. You must experiment with the actual workloads to understand your application needs on the target platform.

It is recommended to design your application context to time share multiple modalities, that is, to enable different modalities at different time. The user gets to enjoy different interactions thus keeps the user engaged. Furthermore, the user is less likely to get muscle fatigue by doing a single interaction too long. To do so, enable multiple modalities at initialization and then pause/resume the modalities, as illustrated in Example 6. The alternative is to close the [SenseManager](#) pipeline and reinitialize for each modality interaction, which works best if the application context switch is long enough to allow camera close and reopen.

**Example 6: Pause Face Computation**

```
C++
```
Coordinate Systems

Finally, it is important to understand the coordinate system that each modality operates when there is a need to combine different modality data together. See Coordinate Systems for the coordinate system definitions. The following table shows the coordinate systems that the modalities operate on for reference:

<table>
<thead>
<tr>
<th>Coordinate System Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Face Tracking (2D)</td>
</tr>
<tr>
<td>Use the color image coordinates.</td>
</tr>
<tr>
<td>Face Tracking (3D)</td>
</tr>
<tr>
<td>Use the camera coordinate system (depth sensor origin).</td>
</tr>
<tr>
<td>Hand Tracking</td>
</tr>
<tr>
<td>Use the camera coordinate system (depth sensor origin).</td>
</tr>
<tr>
<td>Object Tracking</td>
</tr>
<tr>
<td>Use the camera coordinate system (color sensor origin).</td>
</tr>
</tbody>
</table>

For example, you may use hand tracking to identify the hand position and use object tracking to locate a virtual object. You must use the Projection interface to map the coordinates of the output data to the same coordinate system, or you would find that the hand and the virtual object are at two distinct places. Example 7 shows how to do color/depth coordinates conversion.

Example 7: Color/Depth Coordinates Conversion

(C++)
// device is a PXCCapture::Device instance
PXCPProjection *projection=device->CreateProjection();

// Convert from depth(u,v,z) to color(i,j)
{
  PXCPPoint3DF32 depth = {u, v, z};
  PXCPPointF32 color;
  projection->MapDepthToColor(1, &depth, &color);
  i = color.x;
  j = color.y;
}

// Convert from color(i,j) to depth(u,v)
// sample is a PXCCapture::Sample instance
{
  PXCPPointF32 color = {x, y};
  PXCPPointF32 depth;
  projection->MapColorToDepth(sample->depth, 1, &color, &depth);
  u = depth.x;
  v = depth.y;
}

// Clean up
projection->Release();

(C#)

// device is a PXCMCapture.Device instance
PXCMProjection projection=device.CreateProjection();

// Convert from depth(u,v,z) to color(i,j)
{
  PXCMPoint3DF32[] depth=new PXCMPoint3DF32[1];
  depth[0].x=u;
  depth[0].y=v;
  depth[0].z=z;
  PXCMPointF32[] color;
  projection.MapDepthToColor(depth, out color);
  i = color[0].x;
  j = color[0].y;
}

// Convert from color(i,j) to depth(u,v)
// sample is a PXCMCapture.Sample instance
{
  PXCMPointF32[] color = new PXCMPointF32[1];
  color[0].x=x;
  color[0].y=y;
  PXCMPointF32[] depth;
  projection.MapColorToDepth(sample.depth, color, out depth);
  u = depth[0].x;
  v = depth[0].y;
}

// Clean up
projection.Dispose();

(Java)
// device is a PXCMCapture.Device instance
PXCMProjection projection=device.CreateProjection();

// Convert from depth(u,v,z) to color(i,j)
{
    PXCMPoint3DF32[] depth=new PXCMPoint3DF32[1];
    depth[0]=new PXCMPoint3DF32();
    depth[0].x=u;
    depth[0].y=v;
    depth[0].z=z;
    PXCMPointF32[] color=new PXCMPoint3DF32[1];
    PXCMPointF32[0]=new PXCMPoint3DF32();
    projection.MapDepthToColor(depth, color);
    i = color[0].x;
    j = color[0].y;
}

// Convert from color(i,j) to depth(u,v)
// sample is a PXCMCapture.Sample instance
{
    PXCMPointF32[] color = new PXCMPointF32[1];
    color[0]=new PXCMPointF32();
    color[0].x=x;
    color[0].y=y;
    PXCMPointF32[] depth=new PXCMPointF32[1];
    PXCMPointF32[0]=new PXCMPointF32();
    projection.MapColorToDepth(sample.depth, color, depth);
    u = depth[0].x;
    v = depth[0].y;
}

// Clean up
projection.close();

(JavaScript)

// Not supported
4.4.4.1 Sharing Device Configuration

The SDK provides limited support of multiple application access usage scenarios:

- Two or more applications access to the same device, using the same or compatible configuration. For example, two applications can co-access the device data if application 1 streams color at 640x480x30fps and application 2 streams color at 640x480x30fps and depth at 640x480x30fps.

- Specially developed SDK applications that can adapt to whatever configuration other applications use.

Here device configuration refers to stream resolutions, frame rates and pixel format, excluding any device properties such as color brightness and hues. Device properties are unmanaged resource. See Sharing Device Properties for recommendations how to manage the device properties.

The SDK introduces the concept of static application and adaptive application to describe different application scenarios and how they interact with each other.

Static Application

A static application does not explicitly handle device configuration change during streaming. (The static application may work with multiple resolutions during initialization.) This is the default setting.

Static applications can configure the I/O device if there are no other static applications in active streaming. The first static application can retrieve a full enumeration of available configurations. The second or any other applications sees only the active configuration used by the first application.

Static applications have precedents over adaptive applications. If a static application is launched after an adaptive application, the static application can set the device configuration, and the adaptive application receives a device configuration change event.

During streaming, static applications can expect steady streams of data without any interruption from other static or dynamic application activities.

Non-SDK applications that access the same device may interrupt streaming. Since static applications do not handle configuration change, such interruption breaks the streaming loop. It is recommended that the application implement an outer loop to reinitialize and re-start streaming, as illustrated in Example 8.

Example 8: Restart Streaming Upon Interruption

(C++)
// pp is a PXCSenseManager instance

// An outer loop to handle resolution change
pxcStatus sts=PXCV_STATUS_STREAM_CONFIG_CHANGED;
while (sts==PXCV_STATUS_STREAM_CONFIG_CHANGED) {
    // SenseManager initialization
    pp->EnableXXX();
    sts=pp->Init();
    if (sts<PXCV_STATUS_NO_ERROR) break;

    // SenseManager streaming
    for (;;) {
        sts=pp->AcquireFrame();
        if (sts<PXCV_STATUS_NO_ERROR) break;
        ...        pp->ReleaseFrame();
    }
    pp->Close();
}

(C#)

// pp is a PXCMSenseManager instance

// An outer loop to handle resolution change
pxcmStatus sts=pxcmStatus.PXCM_STATUS_STREAM_CONFIG_CHANGED;
while (sts==pxcmStatus.PXCM_STATUS_STREAM_CONFIG_CHANGED) {
    // SenseManager initialization
    pp.EnableXXX();
    sts=pp.Init();
    if (sts.IsError()) break;

    // SenseManager streaming
    for (;;) {
        sts=pp.AcquireFrame();
        if (sts.IsError()) break;
        ...        pp.ReleaseFrame();
    }
    pp.Close();
}

(Java)
// pp is a PXCMSenseManager instance

// An outer loop to handle resolution change
pxcmStatus sts=pxcmStatus.PXCM_STATUS_STREAM_CONFIG_CHANGED;
while (sts==pxcmStatus.PXCM_STATUS_STREAM_CONFIG_CHANGED) {
    // SenseManager initialization
    pp.EnableXXX();
    sts=pp.Init();
    if (sts.isError()) break;

    // SenseManager streaming
    for (;;) {
        sts=pp.AcquireFrame();
        if (sts.isError()) break;
        ...
        pp.ReleaseFrame();
    }
    pp.Close();
}

(JavaScript)

// sm is a SenseManager instance

// Event to receive the notification
var onStatus=function (sender, status) {
    if (sender==sm &&
        status==intel.realSense.Status.STATUS_STREAM_CONFIG_CHANGED) {
        // in other thread, trigger a restart of the SenseManager initialization sequence.
    }
}

// During the SenseManager initialization, set the event handler.
sm.onStatus=onStatus;

Adaptive Application

An adaptive application responds to the device configuration change event and adjusts its behaviors accordingly. The changed behavior could be:

- The application may restart with the new configuration.
- The application may pause its operation until a new suitable configuration is available.

It is recommended that any service applications handle device configuration change.

Adaptive applications can set the device configuration if there is no other application competing for the device. Otherwise, the adaptive application must use the active configuration. The application receives the STATUS_CONFIG_CHANGED event when there is a configuration change during streaming.

You can use the SetDeviceAllowProfileChange function to instruct the device to operate in the adaptive mode. The device will return the STATUS_CONFIG_CHANGED status code when there is a configuration change. Example 9 illustrates the concept. For simplicity, the
example simply restarts the streaming loop (as in Example 8) when re-configuration occurs.

Example 9: Adaptive Application

(C++)

```cpp
// pp is a PXCSenseManager instance
class MyHandler: public PXCSenseManager::Handler {
public:
    virtual pxcStatus PXCAPI OnConnect(PXCCapture::Device *device, pxcBool connected) {
        if (connected) device->SetDeviceAllowProfileChange(true);
        return PXC_STATUS_NO_ERROR;
    }
};

// An outer loop to handle resolution change
pxcStatus sts=PXC_STATUS_STREAM_CONFIG_CHANGED;
while (sts==PXC_STATUS_STREAM_CONFIG_CHANGED) {
    // SenseManager initialization
    pp->EnableXXX();

    // Signal the device to operate in the adaptive mode.
    MyHandler handler;
    sts=pp->Init(&handler);
    if (sts<PXC_STATUS_NO_ERROR) break;

    // SenseManager streaming
    for (;;) {
        sts=pp->AcquireFrame();
        if (sts<PXC_STATUS_NO_ERROR) break;
    ...
    pp->ReleaseFrame();
    }
    pp->Close();
}
```

(C#)
// pp is a PXCMSenseManager instance
pxcmStatus OnConnect(PXCMCapture.Device device, Boolean connected) {
    if (connected) device.SetDeviceAllowProfileChange(true);
    return pxcStatus.PXCM_STATUS_NO_ERROR;
}

// An outer loop to handle resolution change
pxcmStatus sts=pxcmStatus.PXCM_STATUS_STREAM_CONFIG_CHANGED;
while (sts==pxcmStatus.PXCM_STATUS_STREAM_CONFIG_CHANGED) {
    // SenseManager initialization
    pp.EnableXXX();

    // Instruct the device to operate in the adaptive mode
    PXCMSenseManager.Handler handler=new PXCMSenseManager.Handler();
    handler.onConnect=OnConnect;
    sts=pp.Init(handler);
    if (sts.IsError()) break;

    // SenseManager streaming
    for (;;) {
        sts=pp.AcquireFrame();
        if (sts.IsError()) break;
        ...
        pp.ReleaseFrame();
    }
    pp.Close();
}

(Java)
// pp is a PXCMSenseManager instance

class MyHandler implements PXCMSenseManager.Handler {
public:
    @override pxcmStatus OnConnect(PXCMCapture.Device device, boolean connected) {
        if (connected) device.SetDeviceAllowProfileChange(true);
        return pxcmStatus.PXCM_STATUS_NO_ERROR;
    }
}

// An outer loop to handle resolution change
pxcmStatus sts=pxcmStatus.PXCM_STATUS_STREAM_CONFIG_CHANGED;
while (sts==pxcmStatus.PXCM_STATUS_STREAM_CONFIG_CHANGED) {
    // SenseManager initialization
    pp.EnableXXX();

    // Instruct the device to operate in the adaptive mode
    MyHandler handler=new MyHandler();
    sts=pp.Init(handler);
    if (sts.isError()) break;

    // SenseManager streaming
    for (;;) {
        sts=pp.AcquireFrame();
        if (sts.isError()) break;
        ...
        pp.ReleaseFrame();
    }
    pp.Close();
}

(JavaScript)

// unsupported
// JavaScript applications are static applications.
4.4.5 Coexisting with Other Applications

In a typical user system, multiple SDK applications may run concurrently competing for the physical I/O devices. For example, an SDK-enabled game application may compete the camera resource with an SDK-enabled background service. In this case, it is often impractical to ask the user to physically close one application before launching the other, especially when the application is a background service that is "invisible" to the user.

The following issues must be resolved:

- During setup, how do multiple applications negotiate a common device configuration?
  
  See [Sharing Device Configuration](#) for details.

- During streaming, the user may switch among applications. How do applications maintain the optimal device settings for the application context?
  
  See [Sharing Device Properties](#) for details.

- How do multiple applications share the platform computation resource?
  
  See [Sharing Platform Computations](#) for details.
4.4.5.1 Sharing Device Properties

The device properties (such as color stream brightness, hue, and sharpness) are unmanaged resource in SDK 0.0.0. The applications must behave responsibly to achieve an optimal platform level user experience.

Use the following recommendations when designing your application:

a) **SDK application in window focus**: Whenever the user switches your application to window focus, you must set (reset) the device properties to what your application must use. Certain device property has side effect. For example, turning on color stream auto exposure may lower the color stream frame rate in the low-light condition. To minimize such side effect, always reset all available device properties, as illustrated in Example 10.

**Example 10: Restore Device Properties Upon Receiving Windows Focus**

(C++)

```cpp
// device is a PXCCapture::Device instance
device->RestorePropertiesUponFocus();
```

(C#)

```csharp
// device is a PXCMCapture.Device instance
device.RestorePropertiesUponFocus();
```

(Java)

```java
// device is a PXCMCapture.Device instance
device.RestorePropertiesUponFocus();
```

(JavaScript)

```javascript
// device is a Device instance
device.restorePropertiesUponFocus().then(function() {
  ...
});
```

Upon initialization, you cannot assume that the device properties are at their default values. Set them to default values explicitly if this is critical for your application. Example 11 shows how to set depth stream options to default values.

**Example 11: Reset Depth Stream Options to Default**

(C++)

```cpp
// device is a PXCCapture::Device instance
device->ResetProperties(PXCCapture::STREAM_TYPE_DEPTH);
```

(C#)

```csharp
// device is a PXCMCapture.Device instance
device.ResetProperties(PXCMCapture.StreamType.STREAM_TYPE_DEPTH);
```

(Java)

```java
// device is a PXCMCapture.Device instance
device.ResetProperties(PXCMCapture.StreamType.STREAM_TYPE_DEPTH);
```
(JavaScript)

```javascript
// device is a Device instance
device.resetProperties(intel.realsense.StreamType.STREAM_TYPE_DEPTH).then(
    function() {
        ....
    }
);
```

b) **SDK application out of window focus**: Do not change any device properties while your application is not in window focus. Doing so may impact the quality/performance of the SDK application in focus. You may read the device properties to determine if they are still suitable for your algorithms. If the answer is negative, turn off your algorithms.

c) **SDK background applications or services**: Any SDK background application should not change the device properties unless there are other mechanisms to ensure that there is no active SDK application at the moment. In certain cases, for example, an authentication application may need to access to the camera exclusively for a period of time. Do so by starting a foreground window that takes the window focus. Then set the device properties. This is similar to the behavior of any regular SDK application that is in window focus.
4.4.5.2 Sharing Platform Computations

Although the SDK is optimized, there are still non-trivial workloads in module algorithm processing. The applications must behave responsibly so that the overall platform experience is not compromised. The recommendation is as follows concerning three types of SDK applications:

a) **SDK application in window focus**: Any SDK application with its window in focus can assume priority in computing resources. The application can use any SDK resources for the best user experience.

b) **SDK application out of window focus**: If the SDK application loses its window focus, the application should behave responsibly and yield computation resources to any foreground application in focus. The application should pause any non-trivial SDK module processing, as illustrated in Example 12.

Example 12: Pause Face Computation

(C++)

```cpp
// pp is a PXCSenseManager instance.
bool pause=IsWindowFocusLost();
pp->PauseFace(pause);
```

(C#)

```csharp
// pp is a PXCMSenseManager instance.
Boolean pause=IsWindowFocusLost();
pp.PauseFace(pause);
```

(Java)

```java
// pp is a PXCMSenseManager instance.
boolean pause=IsWindowFocusLost();
pp.PauseFace(pause);
```

(JavaScript)

```javascript
// fm is a FaceModule instance

window.onfocus=function() {
    fm.pause(false);
}

window.onblur=function() {
    fm.pause(true);
}
```

c) **SDK background applications or services**: Any SDK background applications or services must be specifically designed such that they are in the idling state most of the time and only become active when certain designated events trigger. When they are in the idling state, they do not consume significant computation. When they become active, they perform their functions and then go back to the idle state.
4.4.6 Working with Screen Rotation

The SDK algorithms assume that the Intel® RealSense™ 3D camera installs at the upper portion of the display panel, i.e., rotation degree 0°. The algorithms are trained correspondingly. The user should normally use the camera in this position for best result.

In reality, it is possible that the user rotates the display panel of his/her tablet or laptop that supports screen rotation. For example, the user may rotate the screen 180° for the "tent" position, or 90°/270° for the portrait mode. Your application must anticipate such cases and respond accordingly.

💡 You can use the standard OS mechanism to detect screen orientation and rotation. See details here as a start.

Screen rotation impacts the following aspects that the SDK works with:

- **Field of View**: The camera field of view is changed. For example, in the portrait modes, the camera horizontal field of view is narrowed and the vertical field of view is expanded. You may find that the object you intend to interact with is more likely to go out of the camera field of view, which may lead to unacceptable interaction.

- **View Angle**: With different rotations, the camera views the object at different view angles. Imagine that you make a fist in front of the camera. If the camera is at the top of the display panel, the camera sees the front and top portions of your fist. The bottom portion of your fist is obscured. If the camera is then moved to the bottom of the display panel, the camera now sees the front and bottom portions of your fist. Depending on the algorithms, such view angle change may impact the SDK algorithm output.

Certain SDK algorithms are not affected by screen rotations. See Table 7 for the list of algorithms and their working state against different screen rotations. If you use any of the SDK algorithms sensitive to screen rotation, you need to instruct the user for the "correct" camera position.

<table>
<thead>
<tr>
<th></th>
<th>Rotation 0°</th>
<th>Rotation 90°</th>
<th>Rotation 180°</th>
<th>Rotation 270°</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D Scanning</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>User Segmentation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Object Tracking</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Blob Tracking</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Hand Tracking</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Hand Gesture</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Face Tracking</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Scene Perception</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Enhanced Photography</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Emotion Detection  √

Table 7: Algorithms That Work With Screen Rotation
4.4.7 Managing Power and Performance

Managing power and performance is critical to deliver the best user experience. The SDK uses a simple concept to help manage your application power and performance:

- You can explicitly set the power state: performance or battery. The performance mode instructs the algorithms and the camera to run as best experience as the platform can provide. The battery mode yields the control to the algorithms or the camera to determine if there is chance to save power, by lowering the frame rate, or using a different algorithms etc.

- In the battery mode, the algorithm(s) may suggest the power state. For example, the hand module may suggest (after some computation) that there is no hand in the scene. Thus it is safe to go into the battery mode. The face module may suggest to go into the battery mode if there is no face in the scene. If any of the sources suggests the performance mode, the SDK is in the performance mode. Otherwise, the SDK is in the battery mode.

The power state is managed by the `PowerState` interface. Example 13 shows an example of how to set the power state to be in the performance mode.

The default state is the battery mode. You want to keep the system in the battery mode as much as possible. Changing the state to the performance mode only briefly when you need the best interaction experience.

Example 13: Manage Power State and Inactivity Interval

[C++]
```cpp
// ss is a PXCSession instance.
PXCPowerState *ps=ss->CreatePowerManager();

// Set the power state
ps->SetState(PXCPowerState::STATE_PERFORMANCE);

// Set the inactivity interval.
ps->SetInactivityInterval(5);

// Clean up
ps->Release();
```

[C#]
```csharp
// ss is a PXCMSession instance.
PXCMPowerState ps=ss.CreatePowerManager();

// Set the power state
ps.SetState(PXCMPowerState.State.STATE_PERFORMANCE);

// Set the inactivity interval.
p->SetInactivityInterval(5);

// Clean up
ps.Dispose();
```
(Java)

```java
// ss is a PXCMSession instance.
PXCMPowerState ps=ss.CreatePowerManager();

// Set the power state
ps.SetState(PXCMPowerState.State.STATE_PERFORMANCE);

// Set the inactivity interval.
ps.SetInactivityInterval(5);

// Clean up
pp.close();
```

(JavaScript)

```javascript
// unsupported
```
4.4.8 Working with Platform Analyzer

Platform Analyzer (part of the Intel® Graphic Performance Analyzers suite) is targeted for performance analysis of applications that use a Graphics Processing Unit (GPU) for rendering, video processing, and computations. Platform Analyzer collects real-time trace data during the application run and provides information on the code execution on the various CPU and GPU cores in your system. The tool infrastructure automatically aligns clocks across all cores in the entire system so that you can analyze some CPU-based workloads together with GPU-based workloads within a unified time domain.

Use the Platform Analyzer to:

- Explore GPU usage and analyze a software queue for GPU engines at each moment of time.
- Analyze GPU usage per DMA packet on a software queue.
- Analyze Microsoft® DirectX® API calls (draw calls, buffer locks, resource updates, presents).
- Identify how effectively your application uses OpenCL™ kernels.
- Analyze Intel® RealSense™ applications and correlate the Intel RealSense SDK tasks execution with the CPU and GPU usage.
- Correlate CPU and GPU activity and identify whether your application is GPU or CPU bound.
- Explore your application performance for user tasks created with Intel ITT API.
- Identify GPU and CPU application frame rate and how it depends on vertical synchronization.
- Explore the performance of your application per selected GPU metrics over time.
Figure 7: The Platform Analysis Flow
4.4.8.1 Setting Up an Application for Analysis

For platform analysis, you don't need to modify your code or special libraries. For advanced analysis, you may enable the VERBOSE tracking level for your application by using the Intel® RealSense™ SDK information viewer, i.e., the sdk_info tool.

Selecting an Application for Analysis

1. Launch Graphics Monitor from the Windows* Start menu or Windows* 8 Apps view.

   The Graphics Monitor icon shows up in the taskbar notification area:

[Image]

2. Double-click the Graphics Monitor icon, or click it and select Analyze Application... from the menu.

   The Analyze Application dialog box opens.

3. In the Windows Desktop Applications pane, click the Browse button in the Command Line field and navigate to your application. If required, enter any command-line options for running the application to be analyzed.
4. (Optionally,) in the Working Folder field, type the directory in which the application should run. This is useful for applications that use a launcher or mission packs in a different directory than the application files.

5. Click the Run button to launch your application.
4.4.8.2 Configuring and Running the Analysis

1. Click the Graphics Monitor icon and select System Analyzer from the menu.

The System Analyzer opens the Connection dialog box with the <This Machine> connection option pre-selected.

2. Click Connect.

System Analyzer displays options for analysis: System Profiling and User Applications.

3. In the User Applications list, select your running application.

The System Analyzer displays a real-time monitor for application and system behavior:

Use the following controls to manage the System Analyzer tool:

- **Toolbar**
  - **Back** - return to the Connection dialog box.
  - **CSV Export** - export metric values to the CSV file to post-process the analysis of your application.
  - **Trace Capture** - capture a trace.
- **Pause/Continue** - pause the application run, or continues if Pause was pressed.

**Frame Per Second pane**

This pane displays:

- the current FPS (frames per second) value
- high(max) and low(min) FPS over one second of application run time

**Metrics Control pane**

This pane displays a group of hardware, driver, and application metrics used for analyzing graphics application performance. For different systems and workloads, different metrics may be available. All metrics are subdivided into logical groups as described in the Metrics List topic in the Graphics Performance Analyzers Help.

**Metrics Chart pane**

This pane contains up to 16 charts that visualize values for various metrics collected by the System Analyzer. By default, two charts are displayed: one containing the Frame Time metric, and another one is empty. To update a chart list with a new metric, select this metric in the Metrics Control pane and drag it to the Metrics Chart pane.

1. Click the Trace Capture button to capture the trace for platform analysis.

The trace capture file *.gpa_trace.amplxe is created.

The default trace duration is 5 seconds. You may change this value by editing the Trace duration setting in the Profiles dialog box >Tracing tab. To access this dialog box, click the Graphics Monitor icon and select Profiles... from the menu.

2. Close the System Analyzer window.
4.4.8.3 Viewing Platform Analysis Data

1. Click the Graphics Monitor icon and select Platform Analyzer from the menu. The Platform Analyzer window opens.

2. From the navigation pane on the left, select the trace file you captured and double-click it to open your trace capture file in the Platform Analysis view.

   In the Platform Analyzer window, you can also click the menu button and select the Result... option to navigate to a Platform Analysis trace and open it.

Platform Analyzer graphically represents platform analysis data over time:

<table>
<thead>
<tr>
<th>Use This</th>
<th>To Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Navigator</td>
<td>open the Project Navigator pane to navigate between trace result files captured with the Platform Analyzer. Data for the selected file shows up in the right pane. The Platform Analyzer window displays only the results that have been created on the current analysis system, with the current version of the product. To</td>
</tr>
</tbody>
</table>
To Do This

- manage data in the Project Navigator, right-click a trace result entry and select the required context menu command.
  - Help - access the product documentation.

2 Timeline toolbar

Navigate the view by zooming in/out the data.

3 GPU Engine

Explore overall GPU usage per GPU engine or packet type at each moment of time. By default, the Platform Analyzer window displays GPU Usage and software queues per GPU engine. Hover over an object executed on the GPU (in yellow) to view a short summary on GPU usage, where GPU Usage is the time when a GPU engine was executing a workload. You can explore the top GPU Usage band in the chart to estimate the percentage of GPU engine utilization (yellow areas vs. white spaces) and options to submit additional work to the hardware.

To view and analyze GPU software queues, select an object (DMA packet) in the queue and the Platform Analyzer highlights the corresponding software queue bounds:

![Platform Analyzer window with GPU software queues highlighted.](image)

Full software queue prevents packet submissions and causes waits on a CPU side in the user-mode driver until there is space in the queue. To check whether such a stall decreases your performance, you may decrease a workload on the hardware and see if there are less waits on the CPU in threads that spawn packets. Another option could be to additionally load the queue by tasks and see whether the queue length increases.

Each DMA packet in the Platform Analyzer window has its own ID that helps
track its life cycle in a software queue. The ID does not correspond to the rendered frames. You may identify where a packet came from by the thread name (corresponding to the name of the module where a thread entry point resides) specified in the tooltip.

Presents are displayed in a red hatch.

On systems with Intel Processor Graphics, you may select the Packet Type drop-down menu option in the Legend area to explore GPU usage and software queues per DMA packet domain:

![Image of Packet Type drop-down menu in Legend area]

**Thread Lifetime**

Explore CPU utilization by thread. Platform Analyzer provides the thread name as a name of the module where the thread function resides. For example, if you have a `myFoo` function that belongs to `MyMegaFoo.dll`, the thread name is displayed as `MyMegaFoo.dll`. This approach helps easily identify the location of the thread code producing the work displayed on the timeline.

Hover over a context switch area to see the details on its duration, reason, and affected CPU. Dark-green context switches show time slices when a thread was busy with a workload while light-green context switch objects show areas where a thread was waiting for a synchronization object. Gray areas show inactivity periods caused by preemption when the operating system task scheduler switched a thread off a processor to run another, higher-priority thread.

Correlate CPU and GPU usage and estimate whether your application is CPU or GPU bound. GPU Engines Usage bars show DMA packets on CPU threads originating GPU tasks. The bars are colored according to the type of used GPU engine (yellow bars in the example below correspond to the Render and
GPGPU engine). If the GPU Engine area of the Platform View shows aggregated GPU usage for all threads and processes in the system, the GPU Engines Usage bars in the Thread Lifetime area show GPU engine utilization by a particular thread.

You can also zoom in to identify user-defined tasks or Intel RealSense SDK tasks executed at particular time frames and correlate this data with the GPU usage at the same time:

**Platform Metrics**

Correlate the data on GPU and CPU activity per thread and your application performance per GPU metrics selected from the System Analyzer window.

**Statistics pane**

Drag and drop to select a range of interest on the timeline. The Statistics pane is updated to synchronize with your selection. Analyze the statistics per metrics to understand whether the workload in the selected time range GPU-bound and identify hotspot objects. Depending on the objects included into the selected range, the Statistics pane may display the following data:

- **GPU Usage**
  - **GPU Time** shows an amount of time used by GPU engines (total and per engine).
  - **Queue Time** shows an amount of time spent in a software queue (total and per engine).
<table>
<thead>
<tr>
<th>Use This</th>
<th>To Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td>o Tasks, where tasks are ITT API calls, Intel RealSense SDK calls and so on.</td>
<td></td>
</tr>
</tbody>
</table>

- Task Time shows an amount of time spent within a task (total and per task).
- Average Task Time shows an average amount of time spent within a task (total and per task).
- Task Count shows a number of tasks in the selected time region (total and per task).

Legend controls
Filter in/out any type of data presented in the timeline by selecting/deselecting corresponding check boxes and drop-down menu options.

Select the VSync check box to display markers for vertical synchronization. Use this data to identify dependencies between GPU frames and VSync events. Depending on your hardware, this option may not show up in the Platform View for an application that does not use vertical synchronization. If vertical synchronization is not enabled for your application, you can use the Platform View to identify a real frame rate for your code. If your application uses vertical synchronization, you can select the VSync timeline option, estimate the correlation between VSync events and application frames, identify frames missing VSync events and explore possible reasons.

Context menus
Right-click and select commands to navigate the data by zooming in/filtering in selected time ranges/objects and adjust the view (band height and timescale) according to your preferences.
4.4.8.4 Advanced Tracking Operations

By default, the Platform Analyzer uses the TRACE level to log the data during analysis. This level of detail is sufficient to analyze the CPU and GPU usage of an Intel® RealSense™ application. Though, if you need to analyze internal Intel RealSense SDK tasks executed during the application run, you may enable an advanced VERBOSE logging level. To do that, use the sdk_info tool and select the VERBOSE option from the Level drop-down menu:

![SDK Information Dialog](image)

For detailed documentation on all analysis options provided with the Platform Analyzer, see the Graphics Performance Analyzers Help.

See Also

The sdk_info tool: Logging Control
4.4.9 Privacy Notice Guidelines

The SDK license contains a set of requirements for meeting end user privacy expectations in your application. This section provides some additional information to assist in meeting these expectations.

There are several types of data accessed by or produced by the SDK that could potentially contain personally identifiable information (PII). Some examples (not an exclusive list) are as follows:

- Raw color or depth data from the camera
- Raw microphone data
- Face recognition data
- Mesh model from 3D scanning

The SDK license requires that your application provide notice to the user when one or more of these data types are accessed, either by the SDK or by your application.

If the PII data is accessed by the SDK but not by the application, inform the user that your application does not have access to the PII data, which will increase the user confidence to leave the camera/microphone on. For example, tell the user that "The application will activate your camera for gesture control. By design, the application has no access to any camera images."

If the PII data is accessed by your application, inform the user what your application does with the PII data. Where appropriate, obtain user consent. Some examples (not an exclusive list) are as follows:

- Store the data on the local device
- Transmit the data over the internet
- Store the data on a remote server
- Share the data with a third party

In your notice, use descriptive language to explain the PII data types so the user can understand why it is of concern. Some examples (not an exclusive list) are as follows:

- Raw Camera Data: The application will have access to the raw picture data coming from the camera. This includes a color image and a depth image.
- Raw Microphone Data: The application will have access to the raw sound data coming from the microphone.
- Face Recognition: The application will identify you based on your face. Your identity will be stored as your name and user ID.
• 3D Mesh Model: The application will create a 3D model of your face that can be used to print or create a game avatar.