# TouchSense® Android Developer Kit for DRV2605 Programming Guide

Version 1.1

# Contents

# 1 Introduction

Immersion’s TouchSense technology enables haptic feedback for mobile computing devices, mobile handsets, touch screens, touch surfaces, and buttons.

Haptic feedback, also known as tactile feedback, touch feedback, and vibro-tactile feedback, allows for a more intuitive, engaging, and natural experience for the user. Haptics can improve the user interface by making on-screen buttons feel like they press and release; by improving the usability of sliders, scrolling lists, and list end stops; and by providing attention-getting tactile components for alarms and error conditions. TouchSense tactile sensations combine well with audio feedback and graphics to create a more immersive, complete, and intuitive multisensory experience.

The TouchSense Android Developer Kit for DRV2605 provides the components that developers need in order to benefit from the haptic capabilities present in TouchSense-enabled mobile handsets. The Developer Kit includes header files, libraries, sample application code, and documentation to assist application developers in using the TouchSense Solution for DRV2605.

This *Programming Guide* provides detailed information on how to use Immersion’s Haptic Java API and Haptic C API, which are part of the TouchSense Solution for DRV2605. This document will assist application developers in working through the various software development considerations.

# 2 Solution Overview

The TouchSense Solution for DRV2605 enables haptics in mobile handsets. TouchSense implementations add tactile feedback as illustrated in Figure 1. A user interacts with the device and tactile sensations (outputs) are generated by TouchSense technology. Tactile output is produced through actuators, electronics, mechanical mounting, and software.

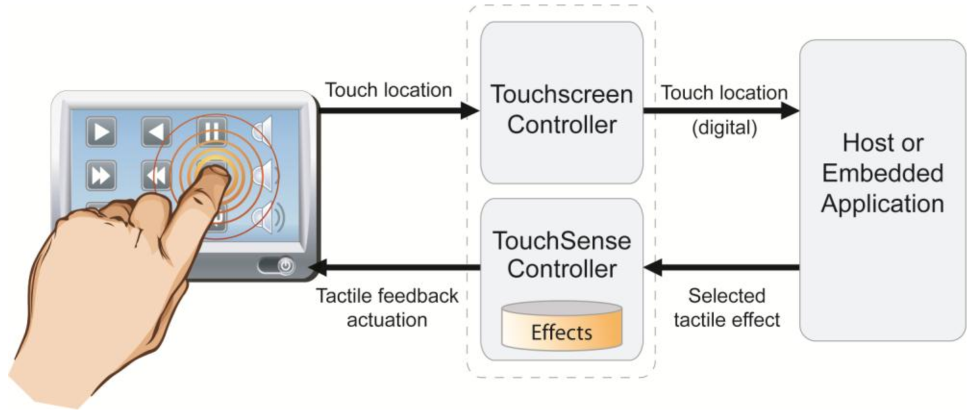

**Figure 1: TouchSense Solution for DRV2605 Architecture (typical)**

- **Mobile Handset** – an easily-portable communication device.
- **Host or Embedded Application** – the host or embedded computer receives input from a user or other source. If the host or embedded application determines that tactile feedback is appropriate, it commands the TouchSense controller to send a TouchSense effect or vibration file to the actuator hardware. Applications normally use software components of the TouchSense Solution for DRV2605 to handle the details of communicating with the TouchSense controller.
- **TouchSense Controller** – Comprised of a tactile feedback microprocessor incorporating Immersion proprietary control firmware and the actuator drive circuit that converts host or embedded computer commands into commands to drive the actuator hardware in playing the tactile effect.
- **Actuator Hardware*** – Off-the-shelf components that generate vibration of the device.

**These components must be purchased or manufactured by other providers. Immersion does not supply these components.*

The software components of the TouchSense Solution for DRV2605 support Android™ applications written in Java, and Windows® applications written in C. Windows is supported for evaluation purposes and is not normally a target platform for the Solution. Figure 2 gives an overview of the Solution.

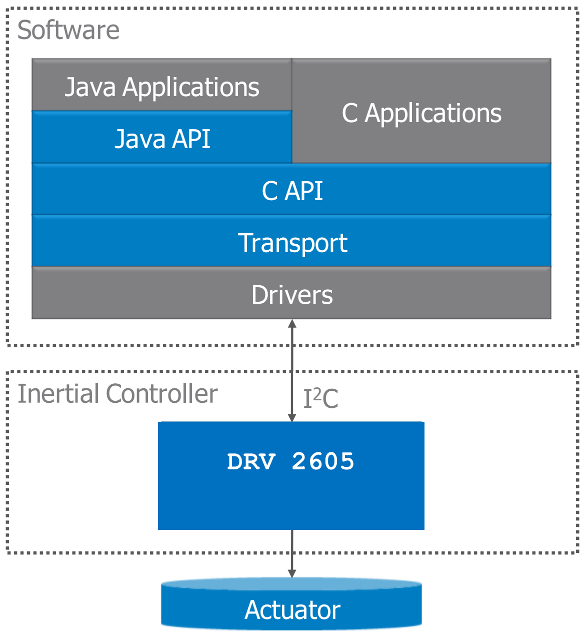

**Figure 2: TouchSense Solution for DRV2605 Overview**

Android applications written in Java use a Haptic Java API that is implemented on top of the Haptic C API.

C applications call functions in the Haptic C API to command the hardware to play haptic effects. The C API has the following desirable properties:

- The Haptic C API handles calls from multiple applications. The C API runs in each application’s process space.
- The Haptic C API is platform-independent. Applications use the C API the same way on different platforms.

# 3 Using the Java API

The Haptic Java API implemented by the **Haptic** class uses the Haptic C API. Every function in the C API is reflected by a public static method in the Java API. To use the Java API:

- Call the **initialize** method once, typically when your application initializes.
- Call the **playEffect** method whenever you want to play an effect from your application.
- Call the **playTimedEffect** method whenever you want to play a pre-defined effect from your application, for a specified duration.
- Call the **playEffectSequence** method whenever you want to play an application-defined sequence of effects from your application.
- Call the **terminate** method once, typically when your application gets destroyed.

Whenever you start playing a new haptic effect, if there is a previous effect still playing, the old effect stops playing before the new one starts. If you want to stop any playing effect without starting a new one, call the **stopPlayingEffect** method.

## 3.1 Developing Your Application

### 3.1.1 Initializing the Java API

Applications must call **initialize**, typically during application initialization, before calling any other Haptic Java API method. The **initialize** method establishes communication with the haptic hardware.

The code below shows how an application can call **initialize** to prepare the Haptic Java API for use.

```
    public void myHapticInitialize()
    {
        try
        {
            Haptic.initialize();
        }
        catch (RuntimeException x)
        {
            // handle error, application-specific
        }
    }
```

### 3.1.2 Terminating the Java API

Applications should call **terminate**, typically during application destruction, after calling all other Haptic Java API methods. The **terminate** method frees resources such as dynamically-allocated memory, and releases any connection to the haptic hardware.

The code below shows how an application can call **terminate**.

```
    public void myHapticTerminate()
    {
        try
        {
            Haptic.terminate();
        }
        catch (RuntimeException x)
        {
            // handle error, application-specific
        }
    }
```

## 3.1.3 Playing Single Haptic Effects

The Haptic Java API provides a variety of effects that applications can trigger by calling **playEffect**. Refer to Appendix A for a description of the available effects.

The code below shows how an application can call the **playEffect** method.

```
    public void myHapticPlayEffect(int nEffect)
    {
        try
        {
            Haptic.playEffect(nEffect);
        }
        catch (RuntimeException x)
        {
            // handle error, application-specific
        }
    }
```

## 3.1.4 Playing Timed Haptic Effects

The Haptic Java API provides applications with the ability to play a pre-defined effect for a desired duration, by calling **playTimedEffect**.

The code below shows how an application can call the **playTimedEffect** method.

```
    public void myHapticPlayTimedEffect(int Duration)
    {
        try
        {
            Haptic.playTimedEffect(nDuration);
        }
        catch (RuntimeException x)
        {
            // handle error, application-specific
        }
    }
```

## 3.1.5 Playing Application-Defined Patterns of Effects

The Haptic Java API allows applications to play sequences of effects. Applications can set up a buffer containing the desired effect indices along with the desired delays between adjacent effects. The inter-effect delays are specified in multiples of 10ms. If the 7th bit of the byte is set to 1, that element in the buffer will be treated as a delay element. Beware of the following limitations:

- Maximum number of effects and delays in a sequence: 8

Refer to Appendix A for a description of the available effects that can be combined into sequences.

The code in below shows how an application can call **playEffectSequence** to play a pattern of two effects.

```
    public void myHapticPlayEffectSequence2(int nEffect0,
                                            int nDelay0,
                                            int nEffect1,
                                            int nDelay1)
    {
        byte[] aBuffer = { nEffect0, nDelay0, nEffect1, nDelay1 };
        try
        {
            Haptic.playEffectSequence(aBuffer, aBuffer.length);
        }
        catch (RuntimeException x)
        {
            // handle error, application-specific
        }
    }
```

### 3.1.6 Stopping Playing Effects

The code below shows how an application can call **stopPlayingEffect** to stop any effect that may be playing.

```
    public void myHapticStopPlayingEffect()
    {
        try
        {
            Haptic.stopPlayingEffect();
        }
        catch (RuntimeException x)
        {
            // handle error, application-specific
        }
    }
```

### 3.1.7 Enabling and Disabling Audio-to-Haptic

The code below shows how an application can call **isAudioHapticEnabled** and **setAudioHapticEnabled** to check, enable or disable Audio-to-Haptic.

```
    public void myHapticToggleAudioToHaptic()
    {
        try
        {
            if (!Haptic.isAudioHapticEnabled())
            {
                Haptic.setAudioHapticEnabled(true);
            } else
            {
                Haptic.setAudioHapticEnabled(false);
            }
        }
        catch (RuntimeException x)
        {
            // handle error, application-specific
        }
    }
```

### 3.1.8 Running Diagnostic

The code below shows how an application can call **runDiagnostic** to tell DRV2605 chip to perform diagnostic on the actuator.

```
    public void myHapticRunDiagnostic()
    {
        int result;
        try
        {
            result = Haptic.runDiagnostic();
        }
        catch (RuntimeException x)
        {
            // handle error, application-specific
        }
    }
```

### 3.1.9 Retrieving DRV260x Chip Revision

The code below shows how an application can call **getChipRevision** to tell the driver to retrieve the revision number of the chip.

```
    public void myHapticGetChipRevision()
    {
        int revision;
        try
        {
            revision = Haptic.getChipRevision();
        }
        catch (RuntimeException x)
        {
            // handle error, application-specific
        }
    }
```

### 3.1.10 Retrieving DRV260x Chip ID

The code below shows how an application can call **getDeviceID** to tell the driver to retrieve the ID of the chip.

```
    public void myHapticGetChipRevision()
    {
        int device;
        try
        {
            device = Haptic.getDeviceID();
        }
        catch (RuntimeException x)
        {
            // handle error, application-specific
        }
    }
```

## 3.2 Building Your Application

The Eclipse IDE automatically rebuilds your application if necessary prior to running it.

## 3.3 Testing and Debugging Your Application

All methods of the Haptic Java API report errors by throwing a **RuntimeException**. The detail message of the **RuntimeException** is a string representation of the Haptic C API error code; for example, “HAPTIC_E_FAIL”. Table 1 describes the detail messages.

**Table 1: Haptic Java API** RuntimeException **Detail Messages**

| Detail Message | Description |
|---|---|
| “HAPTIC_E_FAIL” | General error not covered by more specific detail messages. |
| “HAPTIC_E_INTERNAL” | Internal error. Should not arise. Possibly due to a programming error in the Haptic C API or Haptic Transport. |
| “HAPTIC_E_SYSTEM” | System error. A system call returned an error. |
| “HAPTIC_E_MEMORY” | Memory allocation error. Unable to allocate required memory. |
| “HAPTIC_E_ARGUMENT” | Null or invalid argument. |
| “HAPTIC_E_ALREADY_INITIALIZED” | The Haptic Java API is already initialized. The application has already called the **initialize** method without an intervening call to the **terminate** method. |
| “HAPTIC_E_NOT_INITIALIZED” | The Haptic Java API is not initialized. The application must first call the **initialize** method. |
| “HAPTIC_E_UNSUPPORTED” | The Haptic Java API is not installed on this device. The application will not be able to play any effect. |

## 3.4 Deploying Your Application

Because the Haptic Java API and dependent libraries may be tailored to specific devices, those libraries should already be pre-installed by the manufacturer on supported devices; therefore, you should not distribute any Java API libraries with your application.

# 4 Using the C API

To use the Haptic C API:

- Call the **ImHapticInitialize** function once, typically when your application initializes.
- Call the **ImHapticPlayEffect** function whenever you want to play an effect from your application.
- Call the **ImHapticPlayEffectSequence** function whenever you want to play a sequence of effects from your application.
- Call the **ImHapticGetDevID** function whenever you want to get the ID of the current chip.
- Call the **ImHapticGetChipRev** function whenever you want to get the revision of the current chip.
- Call the **ImHapticRunDiagnostic** function whenever you want to get the chip to perform diagnostic procedures.
- Call the **ImHapticTerminate** function once, typically when your application terminates.

Whenever you start playing a new haptic effect, if there is a previous effect still playing, the old effect stops playing before the new one starts. If you want to stop any playing effect without starting a new one, call the **ImHapticStopPlayingEffect** function.

The following sub-sections provide additional information about using the Haptic C API, including examples of calling each function, and best-practice suggestions to aid in development.

## 4.1 Developing Your Application

### 4.1.1 Initializing the C API

Applications must call **ImHapticInitialize**, typically during application initialization, before calling any other Haptic C API function. **ImHapticInitialize** establishes communication with the haptic hardware.

The code below shows how an application can call **ImHapticInitialize** to prepare the Haptic C API for use.

```
void MyHapticInitialize()
{
    HapticResult res;

    res = ImHapticInitialize();
    if (HAPTIC_FAILED(res))
    {
        /* handle error, application-specific */
        return;
    }
}
```

### 4.1.2 Terminating the C API

Applications should call **ImHapticTerminate**, typically during application termination, after calling all other Haptic C API functions. **ImHapticTerminate** frees resources such as dynamically-allocated memory, and releases any connection to the haptic hardware.

The code below shows how an application can call **ImHapticTerminate**.

```
void MyHapticTerminate()
{
    HapticResult res;

    res = ImHapticTerminate();
    if (HAPTIC_FAILED(res))
    {
        /* handle error, application-specific */
        return;
    }
}
```

### 4.1.3 Playing Single Haptic Effects

The Haptic C API provides a variety of effects that applications can trigger by calling **ImHapticPlayEffect**. Refer to Appendix A for a description of the available effects.

The code below shows how an application can call **ImHapticPlayEffect**.

```
void MyHapticPlayEffect(HapticInt nEffect)
{
    HapticResult res;

    res = ImHapticPlayEffect(nEffect);
    if (HAPTIC_FAILED(res))
    {
        /* handle error, application-specific */
        return;
    }
}
```

### 4.1.4 Playing Timed Haptic Effects

The Haptic C API provides applications with the ability to play a pre-defined effect for a desired duration, by calling **ImHapticPlayTimedEffect**.

The code below shows how an application can call **ImHapticPlayTimedEffect**.

```
void MyHapticPlayTimedEffect(HapticInt nDuration)
{
    HapticResult res;

    res = ImHapticPlayTimedEffect(nDuration);
    if (HAPTIC_FAILED(res))
    {
        /* handle error, application-specific */
        return;
    }
}
```

### 4.1.5 Playing Application-Defined Patterns of Effects

The Haptic C API allows applications to play sequences of effects. Applications can set up a buffer containing the desired effect indices along with the desired delays between adjacent effects. The inter-effect delays are specified in multiples of 5ms. If the 7th bit of the byte is set to 1, that element in the buffer will be treated as a delay element. Beware of the following limitations:

- Maximum number of effects and delays in a sequence: 8

Refer to Appendix A for a description of the available effects that can be combined into sequences.

The code below shows how an application can call **ImHapticPlayEffectSequence** to play a pattern of two effects.

```
void MyHapticPlayEffectSequence2(HapticInt  nEffect0,
                                 HapticInt  nDelay0,
                                 HapticInt  nEffect1,
                                 HapticInt  nDelay1)
{
    HapticUInt8 aBuffer[4];
    HapticResult res;

    aBuffer[0] = nEffect0;
    aBuffer[1] = nDelay0;
    aBuffer[2] = nEffect1;
    aBuffer[3] = nDelay1;

    res = ImHapticPlayEffectSequence(aBuffer, 4);
    if (HAPTIC_FAILED(res))
    {
        /* handle error, application-specific */
        return;
    }
}
```

### 4.1.6 Stopping Playing Effects

The code below shows how an application can call **ImHapticStopPlayingEffect** to stop any effect that may be playing.

```
void MyHapticStopPlayingEffect()

{
    HapticResult res;

    res = ImHapticStopPlayingEffect();
    if (HAPTIC_FAILED(res))
    {
        /* handle error, application-specific */
        return;
    }
}
```

### 4.1.7 Enabling and Disabling Audio-to-Haptic

The code below shows how an application can call **ImAudioHapticGetStatus**, **ImAudioHapticEnable** and **ImAudioHapticDisable** to check, enable or disable Audio-to-Haptic.

```
void MyHapticToggleAudioToHaptic()
{
    HapticResult res;
    int enabled;

    res = ImAudioHapticGetStatus(&enabled);
    if (HAPTIC_FAILED(res))
    {
        /* handle error, application-specific */
        return;
    }

    if (!enabled)
    {
        res = ImAudioHapticEnable();
    } else
    {
        res = ImAudioHapticDisable();
    }
    if (HAPTIC_FAILED(res))
    {
        /* handle error, application-specific */
        return;
    }
}
```

### 4.1.8 Running Diagnostic

The code below shows how an application can call **ImHapticRunDiagnostic** to perform diagnostic procedures

```
void MyHapticRunDiagnostic()

{
    HapticResult res;
    int diag_res;

    res = ImHapticRunDiagnostic(&diag_res);
    if (HAPTIC_FAILED(res))
    {
        /* handle error, application-specific */
        return;
    }
}
```

### 4.1.9 Retrieving DRV260x Chip Revision

The code below shows how an application can call **ImHapticGetChipRev** to retrieve the current chip revision

```
void MyHapticGetChipRevision()

{
    HapticResult res;
    int chip_rev;

    res = ImHapticGetChipRev(&chip_rev);
    if (HAPTIC_FAILED(res))
    {
        /* handle error, application-specific */
        return;
    }
}
```

### 4.1.10 Retrieving DRV260x Chip ID

The code below shows how an application can call **ImHapticGetDevID** to retrieve the current chip ID

```
void MyHapticGetChipRevision()

{
    HapticResult res;
    int chip_id;

    res = ImHapticGetChipRev(&chip_id);
    if (HAPTIC_FAILED(res))
    {
        /* handle error, application-specific */
        return;
    }
}
```

## 4.2 Building Your Application

To build your application, ensure that the application project or makefiles are set up to access the Haptic C API header files and library as described in section 4.1.

## 4.3 Testing and Debugging Your Application

All functions of the Haptic C API return a value that can be tested for success or failure. Zero and positive values indicate success; negative values are error codes. Table 2 describes the return status codes.

**Table 2: Haptic C API Return Status Codes**

| Value | Constant Name | Description |
|---|---|---|
| 1 | HAPTIC_S_TRUE | Indicates true condition (no error). |
| 0 | HAPTIC_S_FALSE | Indicates false condition (no error). |
| 0 | HAPTIC_S_OK | No error |
| -1 | HAPTIC_E_FAIL | General error not covered by more specific codes. |
| -2 | HAPTIC_E_INTERNAL | Internal error. Should not arise. Possibly due to a programming error in the Haptic C API or Haptic Transport. |
| -3 | HAPTIC_E_SYSTEM | System error. A system call returned an error. |
| -4 | HAPTIC_E_MEMORY | Memory allocation error. Unable to allocate required memory. |

| Value | Constant Name | Description |
|---|---|---|
| -5 | HAPTIC_E_ARGUMENT | Null or invalid argument. |
| -6 | HAPTIC_E_ALREADY_INITIALIZED | The Haptic C API is already initialized. The application has already called the **ImHapticInitialize** function without an intervening call to the **ImHapticTerminate** function. |
| -7 | HAPTIC_E_NOT_INITIALIZED | The Haptic C API is not initialized. The application must first call the **ImHapticInitialize** function. |
| -9 | HAPTIC_E_UNSUPPORTED | The Haptic C API is not installed on this device. The application will not be able to play any effect. |

## 4.4 Deploying Your Application

Because the Haptic C API and dependent libraries may be tailored to specific devices, those libraries should already be pre-installed by the manufacturer on supported devices; therefore, you should not distribute any C API libraries with your application.

# Appendix A

Table 3 describes the haptic effects that are available for applications to play or combine into sequences.

**Table 3: Effect Library**

| Effect Index | Description | Suggested Usage | | |
|---|---|---|---|---|
| | | Button | Alert | Gesture |
| 1 | Strong Click - 100% | X | | |
| 2 | Strong Click - 60% | X | | |
| 3 | Strong Click - 30% | X | | |
| 4 | Sharp Click - 100% | X | | |
| 5 | Sharp Click - 60% | X | | |
| 6 | Sharp Click - 30% | X | | |
| 7 | Soft Bump - 100% | X | | |
| 8 | Soft Bump - 60% | X | | |
| 9 | Soft Bump - 30% | X | | |
| 10 | Double Click - 100% | X | | |
| 11 | Double Click - 60% | X | | |
| 12 | Triple Click - 100% | X | X | X |
| 13 | Soft Fuzz - 60% | | X | X |
| 14 | Strong Buzz - 100% | | X | X |
| 15 | Long 750 ms Buzz | NA | NA | NA |
| 16 | Long 1000 ms Buzz | NA | NA | NA |
| 17 | Strong Click 1 - 100% | X | | |
| 18 | Strong Click 2 - 80% | X | | |
| 19 | Strong Click 3 - 60% | X | | |
| 20 | Strong Click 4 - 30% | X | | |
| 21 | Medium Click 1 - 100% | X | | |

| Effect Index | Description | Suggested Usage | | |
|---|---|---|---|---|
| | | Button | Alert | Gesture |
| 22 | Medium Click 2 - 80% | X | | |
| 23 | Medium Click 3 - 60% | X | | |
| 24 | Sharp Tick 1 - 100% | X | | |
| 25 | Sharp Tick 2 - 80% | X | | |
| 26 | Sharp Tick 3 – 60% | X | | |
| 27 | Short Double Click Strong 1 – 100% | X | | |
| 28 | Short Double Click Strong 2 – 80% | X | | |
| 29 | Short Double Click Strong 3 – 60% | X | | |
| 30 | Short Double Click Strong 4 – 30% | X | | |
| 31 | Short Double Click Medium 1 – 100% | X | | |
| 32 | Short Double Click Medium 2 – 80% | X | | |
| 33 | Short Double Click Medium 3 – 60% | X | | |
| 34 | Short Double Sharp Tick 1 – 100% | X | | |
| 35 | Short Double Sharp Tick 2 – 80% | X | | |
| 36 | Short Double Sharp Tick 3 – 60% | X | | |
| 37 | Long Double Sharp Click Strong 1 – 100% | X | | |
| 38 | Long Double Sharp Click Strong 2 – 80% | X | | |
| 39 | Long Double Sharp Click Strong 3 – 60% | X | | |
| 40 | Long Double Sharp Click Strong 4 – 30% | X | | |
| 41 | Long Double Sharp Click Medium 1 – 100% | X | | |
| 42 | Long Double Sharp Click Medium 2 – 80% | X | | |
| 43 | Long Double Sharp Click Medium 3 – 60% | X | | |
| 44 | Long Double Sharp Tick 1 – 100% | X | | |
| 45 | Long Double Sharp Tick 2 – 80% | X | | |
| 46 | Long Double Sharp Tick 3 – 60% | X | | |

| **Effect Index** | **Description** | **Suggested Usage** | | |
|---|---|---|---|---|
| | | **Button** | **Alert** | **Gesture** |
| 47 | Buzz 1 – 100% | | X | X |
| 48 | Buzz 2 – 80% | | X | X |
| 49 | Buzz 3 – 60% | | X | X |
| 50 | Buzz 4 – 40% | | X | X |
| 51 | Buzz 5 – 20% | | X | X |
| 52 | Pulsing Strong 1 – 100% | | X | X |
| 53 | Pulsing Strong 2 – 60% | | X | X |
| 54 | Pulsing Medium 1 – 100% | | X | X |
| 55 | Pulsing Medium 2 – 60% | | X | X |
| 56 | Pulsing Sharp 1 – 100% | | X | X |
| 57 | Pulsing Sharp 2 – 60% | | X | X |
| 58 | Transition Click 1 – 100% | | X | X |
| 59 | Transition Click 2 – 80% | | X | X |
| 60 | Transition Click 3 – 60% | | X | X |
| 61 | Transition Click 4 – 40% | | X | X |
| 62 | Transition Click 5 – 20% | | X | X |
| 63 | Transition Click 6 – 10% | | X | X |
| 64 | Smooth Hum (No kick or brake pulse) @80% | | X | X |
| 65 | Smooth Hum (No kick or brake pulse) @65% | | X | X |
| 66 | Transition Hum 3 – 60% | | X | X |
| 67 | Transition Hum 4 – 40% | | X | X |
| 68 | Transition Hum 5 – 20% | | X | X |
| 69 | Transition Hum 6 – 10% | | X | X |
| 70 | Transition Ramp Down Long Smooth 1 – 100 to 0% | | X | X |

| Effect Index | Description | Suggested Usage | | |
|---|---|---|---|---|
| | | Button | Alert | Gesture |
| 71 | Transition Ramp Down Long Smooth 2 – 100 to 0% | | X | X |
| 72 | Transition Ramp Down Medium Smooth 1 – 100 to 0% | | X | X |
| 73 | Transition Ramp Down Medium Smooth 2 – 100 to 0% | | X | X |
| 74 | Transition Ramp Down Short Smooth 1 – 100 to 0% | | X | X |
| 75 | Transition Ramp Down Short Smooth 2 – 100 to 0% | | X | X |
| 76 | Transition Ramp Down Long Sharp 1 – 100 to 0% | | X | X |
| 77 | Transition Ramp Down Long Sharp 2 – 100 to 0% | | X | X |
| 78 | Transition Ramp Down Medium Sharp 1 – 100 to 0% | | X | X |
| 79 | Transition Ramp Down Medium Sharp 2 – 100 to 0% | | X | X |
| 80 | Transition Ramp Down Short Sharp 1 – 100 to 0% | | X | X |
| 81 | Transition Ramp Down Short Sharp 2 – 100 to 0% | | X | X |
| 82 | Transition Ramp Up Long Smooth 1 – 0 to 100% | | X | X |
| 83 | Transition Ramp Up Long Smooth 2 – 0 to 100% | | X | X |
| 84 | Transition Ramp Up Medium Smooth 1 – 0 to 100% | | X | X |
| 85 | Transition Ramp Up Medium Smooth 2 – 0 to 100% | | X | X |
| 86 | Transition Ramp Up Short Smooth 1 – 0 to 100% | | X | X |
| 87 | Transition Ramp Up Short Smooth 2 – 0 to 100% | | X | X |

| Effect Index | Description | Suggested Usage | | |
|---|---|---|---|---|
| | | Button | Alert | Gesture |
| 88 | Transition Ramp Up Long Sharp 1 – 0 to 100% | | X | X |
| 89 | Transition Ramp Up Long Sharp 2 – 0 to 100% | | X | X |
| 90 | Transition Ramp Up Medium Sharp 1 – 0 to 100% | | X | X |
| 91 | Transition Ramp Up Medium Sharp 2 – 0 to 100% | | X | X |
| 92 | Transition Ramp Up Short Sharp 1 – 0 to 100% | | X | X |
| 93 | Transition Ramp Up Short Sharp 2 – 0 to 100% | | X | X |
| 94 | Transition Ramp Down Long Smooth 1 – 50 to 0% | | X | X |
| 95 | Transition Ramp Down Long Smooth 2 – 50 to 0% | | X | X |
| 96 | Transition Ramp Down Medium Smooth 1 – 50 to 0% | | X | X |
| 97 | Transition Ramp Down Medium Smooth 2 – 50 to 0% | | X | X |
| 98 | Transition Ramp Down Short Smooth 1 – 50 to 0% | | X | X |
| 99 | Transition Ramp Down Short Smooth 2 – 50 to 0% | | X | X |
| 100 | Transition Ramp Down Long Sharp 1 – 50 to 0% | | X | X |
| 101 | Transition Ramp Down Long Sharp 2 – 50 to 0% | | X | X |
| 102 | Transition Ramp Down Medium Sharp 1 – 50 to 0% | | X | X |
| 103 | Transition Ramp Down Medium Sharp 2 – 50 to 0% | | X | X |
| 104 | Transition Ramp Down Short Sharp 1 – 50 to 0% | | X | X |

| **Effect Index** | **Description** | **Suggested Usage** | | |
|---|---|---|---|---|
| | | **Button** | **Alert** | **Gesture** |
| 105 | Transition Ramp Down Short Sharp 2 – 50 to 0% | | X | X |
| 106 | Transition Ramp Up Long Smooth 1 – 0 to 50% | | X | X |
| 107 | Transition Ramp Up Long Smooth 2 – 0 to 50% | | X | X |
| 108 | Transition Ramp Up Medium Smooth 1 – 0 to 50% | | X | X |
| 109 | Transition Ramp Up Medium Smooth 2 – 0 to 50% | | X | X |
| 110 | Transition Ramp Up Short Smooth 1 – 0 to 50% | | X | X |
| 111 | Transition Ramp Up Short Smooth 2 – 0 to 50% | | X | X |
| 112 | Transition Ramp Up Long Sharp 1 – 0 to 50% | | X | X |
| 113 | Transition Ramp Up Long Sharp 2 – 0 to 50% | | X | X |
| 114 | Transition Ramp Up Medium Sharp 1 – 0 to 50% | | X | X |
| 115 | Transition Ramp Up Medium Sharp 2 – 0 to 50% | | X | X |
| 116 | Transition Ramp Up Short Sharp 1 – 0 to 50% | | X | X |
| 117 | Transition Ramp Up Short Sharp 2 – 0 to 50% | | X | X |
| 118 | Long buzz for programmatic stopping – 100%, 10 seconds max | X | X | X |
| 119 | Smooth Hum 1 (No kick or brake pulse) – 50%* | | X | X |
| 120 | Smooth Hum 2 (No kick or brake pulse) – 40%* | | X | X |
| 121 | Smooth Hum 3 (No kick or brake pulse) – 30%* | | X | X |
| 122 | Smooth Hum 4 (No kick or brake pulse) – 20%* | | X | X |

| Effect Index | Description | Suggested Usage | | |
|---|---|---|---|---|
| | | Button | Alert | Gesture |
| 123 | Smooth Hum 5 (No kick or brake pulse) – 10%* | | X | X |

*Effects 64, 65, and 119 through 122 do not have kick or brake pulses and may be repeated to obtain longer smooth effects. The sequence may need to be preceded by an effect having a kick pulse in order to activate the haptic actuator.

**Immersion Corporation**
30 Rio Robles
San Jose, CA 95134 USA
www.immersion.com