

ITK Documentation

Release 1.4.2

Impinj

June 12, 2015

CONTENTS

I	Version 1.4.2 Release Documentation	1
1	Contents	3
1.1	<i>Release Notes</i>	3
1.2	<i>Quick Start Guide</i>	3
1.3	<i>Impinj Radio Interface Toolkit for C (ITK-C)</i>	3
1.4	<i>Impinj Radio Interface Toolkit LT for C (ITK-LT-C)</i>	3
1.5	<i>Frequently Asked Questions (FAQ)</i>	3
1.5.1	Release Notes	4
1.5.2	Quick Start Guide	6
1.5.3	Impinj Radio Interface Toolkit LT for C (ITK-LT-C)	9
1.5.4	Frequently Asked Questions (FAQ)	20
1.5.5	Overview	27
1.5.6	IRI Example Programs	30
1.5.7	IRI Configuration Examples	66
1.5.8	Functions	87
1.5.9	Structures	96
1.5.10	Defines	106
1.5.11	Key Codes	119
1.5.12	Regulatory Regions	158
1.5.13	Error Codes	173
1.5.14	Platform and Report Handlers	185
1.5.15	Stored Settings	187
1.5.16	Memory Usage	191

Part I

Version 1.4.2 Release Documentation

The Indy ITK release includes the Impinj Radio Interface Toolkit for C (ITK-C) to communicate with the Indy Reader SiPs, new binary images for the Indy SiP applications, and a GUI tool for loading and controlling the application on the Indy SiPs. This document provides an overview of the ITK release, including Release Notes, Quick Start Guide, a detailed description of the ITK-C, and frequently asked questions.

CONTENTS

Note: Please read the *Release Notes* first

1.1 *Release Notes*

The Release Notes describe changes in the current version of the ITK, as well as historical versions and notes about compatibility.

1.2 *Quick Start Guide*

The Quick Start Guide provides an introduction to the Indy SiP devices, and the ITK. It is a good place to get started for new Indy SiP users.

1.3 *Impinj Radio Interface Toolkit for C (ITK-C)*

The ITK-C documentation explains the IRI host library in detail, including its functions, structures, key codes, and defines, as well as showing example code snippets and programs. This is the main content of the ITK release documentation.

1.4 *Impinj Radio Interface Toolkit LT for C (ITK-LT-C)*

The ITK-LT-C documentation describes the ITK-LT-C, which has lower resource utilization and a smaller feature set than the ITK-C. Common elements of the two ITKs are covered only by the ITK-C documentation, so the ITK-LT-C documentation is significantly smaller.

1.5 *Frequently Asked Questions (FAQ)*

The Frequently Asked Questions answers common queries about the Indy SiPs and about the ITK-C and ITK-LT-C.

1.5.1 Release Notes

Introduction

Information, source code, and binaries provided in this release are subject to change with subsequent releases. Impinj plans to release updates on a quarterly basis going forward. Please visit support.impinj.com for future updates.

Backward Compatibility

RS2000

This release is the first for the RS2000 SiP, so there are no backward compatibility issues.

RS500

There are no backward compatibility issues in this release for the RS500 SiP.

Bug Fixes in the current release

Monza R6 EPC/TID handling

- In previous versions, if a tag's EPC contained a value similar to known TIDs (e.g. First bytes of the EPC equal to E28011) then the tag report would populate the TID field with the EPC and the EPC would be null.

Changes in the current release

- *Multiple RF Modes*
- *Antenna Switching*

High Level Feature Description

- *ITK-LT-C Support*
- *Multiple Regions*
- *Inventory*
- *Select*
- *Stop Action Tag Count*
- *Stop Action Duration*
- *Tag Read*
- *Tag Write*
- *Tag EPC Write*
- *Tag Lock*
- *Tag Blockpermalock*
- *Tag QT*

- *Power Control*
- *FastID*
- *Tag Focus*
- *Regulatory Test Commands CW*
- *Regulatory Test Commands PRBS*
- *Baud Rate Change*
- *Custom Region*
- *Report Field Configuration*
- *User GPIO*
- *Suppress Set Responses*
- *Access Retry Count*
- *Application Update via IRI*
- *Power Management(Standard Idle, Low Latency Idle, Standby, Sleep)*
- *Get Info*
- *Stored Settings*
- *ITK-LT-C*
- *Multiple RF Modes*
- *Antenna Switching*

Not Implemented

The following features are not available in this release:

- *ipj_resume*

Known Defects

The current release does not have any known defects.

Change Log - Previous Releases

v1.1.2.240

- Added support for *ITK-LT-C* – a smaller footprint IRI toolkit
- Added Key for storing third party custom data (*E_IPJ_KEY_GENERIC_DATA*)

v1.0.8.240

- ITK Examples have been simplified and consolidated
- *Stored Settings* feature added

v1.0.6.240

- IRI enumerations have been changed to defines (No functional change and fully compatible with previous versions)
- Get Info Feature added

v1.0.2.240

- Power Management (Standard Idle, Low Latency Idle, Standby, Sleep)

v1.0.0.240

- Renamed index/offset to bank_index/value_index
- Multiple Select Command Support
- IPJ_CLEAR_STRUCT macro to assist in initializing IRI structures
- RSSI in Tag Report reported in centi-dBm

1.5.2 Quick Start Guide

Introduction

This is a step by step guide to help users get started building an embedded system including an Indy Reader SiP, using the ITK Release provided by Impinj.

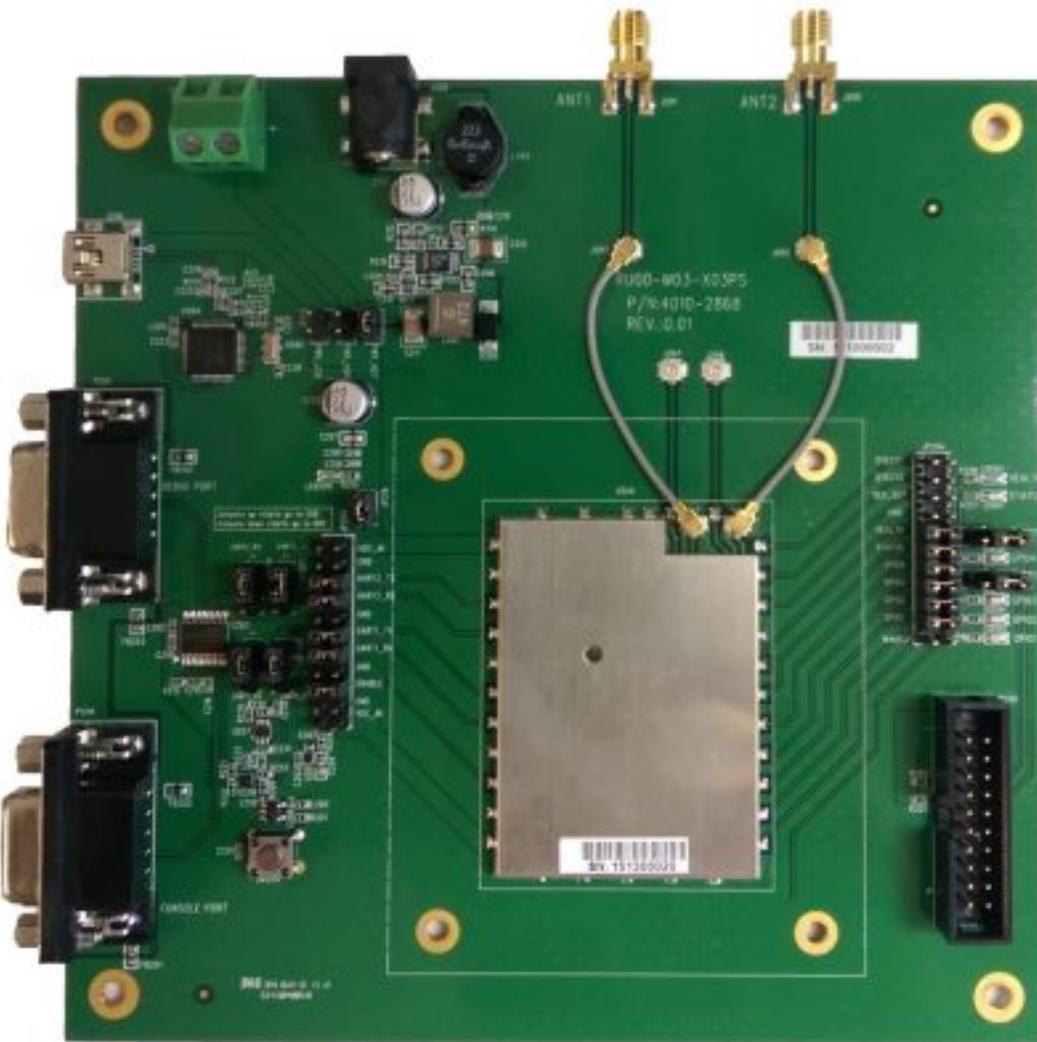
Indy Reader SiPs communicate with a host over a physical full-duplex UART interface. This package, the “Indy ITK Release”, contains collateral to help develop embedded firmware or PC software to communicate with Indy reader SiPs. This version of the release contains two C libraries, IRI and IRI-LT. IRI-LT implements a reduced featureset of IRI, allowing control of an Indy SiP by a host with less resources. Each of these libraries is contained within its own IRI Toolkit (ITK) within the ITK Release. For documentation specific to each of the toolkits, see *ITK-C* and *ITK-LT-C*. These toolkits include source libraries, documentation, and examples.

For more details on the differences between the ITK-C and ITK-LT-C, see the *ITK Differences* section of the FAQ.

The following sections describe the logical progression of steps that Impinj recommends for developing an RS500 reader based RFID solution. Some steps are optional, based on the needs of the user.

Step 1. Get Started with the Indy Reader SiP Development Kits

The Indy reader SiP Development Kits are designed to assist in evaluation and development with Indy Reader SiPs. There is a RS500 and a RS2000 development kit available. If you have just received an Indy SiP Development Kit, and would like to learn how to use it, please see the Quick Start Guide for the kit, which is included in the Release package under `.\Documentation\`, or on the web [here \(for RS500\)](#).



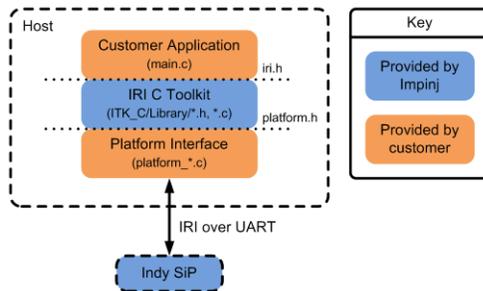
Step 2. Examine the IRI and IRI-LT PC Host Examples

The next step in developing Indy SiP host firmware or software is to take a look at the host PC examples that are shipped along with the ITK Release. These are stored within the Indy ITK Release package at `.\ITK_C\Examples\` and `.\ITK_LT_C\Examples\`. These examples demonstrate a number of functions of the IRI and IRI-LT libraries, and can be run on a host computer connected to an Indy SiP Development Kit via USB. The examples are documented in more detail here: [IRI Examples](#)

Step 3. Develop IRI or IRI-LT Based Embedded Firmware

To develop your own embedded host firmware speaking IRI, you'll need to integrate the IRI or IRI-LT libraries into your embedded firmware project. The diagram below shows the source files required to implement an Indy SiP based system. To implement such a system, create your own C or C++ project, and include `iri.h` or `iri_lt.h`, which are stored in the Release package in `.\ITK_C\Library\` and `.\ITK_LT_C\Library\`. You may also need to include `ipj_util.h`, if you want to take advantage of the API built into that library, which are useful for development and debugging.

Before the included libraries will work, some platform specific code may need to be written. This code implements communication to and from the Indy SiP over UART, and also some other functionality like event timing. This code is contained within the `platform_*.c` source files, which are stored in the Indy ITK Release package in `.\ITK_C\Library\` and `.\ITK_LT_C\Library\`. Source files `platform_linux.c` and `platform_win32.c` are provided to aid development. An empty platform library file, `platform_empty.c` is provided as a template. The text file `PORTING.txt` provides further details on porting device specific code.



Step 4. Develop Embedded Host Firmware using an Indy SiP Development Board

The Indy SiP Development Boards can be used as platforms for early development along with a third party development board. The board has connectors to facilitate easy connection of host UART pins and GPIOs to the pins of the Indy SiP. For more details on how to use the Indy SiP Development Boards in this fashion, see each Indy SiP's Hardware User's Guide, which are included in the Release package under `.\Documentation\`, or on the web [here](#) (for RS500).

Step 5. Integrate an Indy SiP Into a Custom PCB

For help designing a PCB to connect the Indy SiP to a host, see the Indy SiP Hardware User's Guide, which is included in the Release package under `.\Documentation\`, or on the web [here](#) (for RS500).

Step 6. Review Additional Information

For more information, see the following documentation:

- View the rest of the Release documentation such as the *IRI C toolkit* or the *overview*.
- View the rest of the RS500 Hardware documentation, such as the Datasheet and Hardware User's Guide, which are included in the Release package under `.\Documentation\`, or on the web [here](#).
- Browse through the *Frequently Asked Questions*.
- Visit support.impinj.com, where you can submit support tickets and engage in technical discussion.

1.5.3 Impinj Radio Interface Toolkit LT for C (ITK-LT-C)

The ITK-C host library is designed to be run in embedded systems. Embedded systems cover a wide range of platforms and, as such, have a wide variety of system resources available. In some memory constrained applications the system ROM and RAM requirements of the ITK-C may prove to be too great.

To address these applications, the ITK-LT-C has been designed with a subset of the IRI API and features. The result is a small host library with a reduced RAM and ROM footprint with reader functionality that can enable many applications.

For more details on the differences between the ITK-C and ITK-LT-C, see the *ITK Differences* section of the FAQ.

Important: ITK-LT-C should only be used if the the standard ITK-C can not be used due to host memory constraints. The ITK-C is extensible and enables future advanced features.

As a general rule all documentation for the ITK-C applies to ITK-LT-C. The differences are detailed in the sections below.

ITK-LT-C Memory Usage

The following table lists verified ITK-LT-C host platforms and corresponding library sizes.

Notes:

- These results come from library version 1.1.2.240
- **All `config.h` compile time switches were disabled**
 - The `ENABLE_FW_UPDATES` compile time switch increases the library sizes reported below by approximately 750 bytes
- The ITK-C library is comprised of multiple files
- The ITK-LT-C library has been reduced to one file
- Linux, Windows, and OSX ports are available now
- Embedded MCU ports will be included in a future version of the ITK

If you have questions please submit a support ticket at support.impinj.com .

Table 1.1: Library Sizes

Library	Build Machine	Target Platform	Target Architecture	Target Processor	Toolchain	Optimizations	Library Contents	Size (Bytes)
ITK-LT-C	Windows 7 + Cygwin	Windows	32-bit	x86	GCC 4.5.3	For size (-Os)	iri_lt.o	1,412
ITK-LT-C	CentOS 6.5	Linux	32-bit	x86	GCC 4.4.7	For size (-Os)	iri_lt.o	1,248
ITK-LT-C	Apple Mac 10.9.2	OSX	32-bit	x86	GCC (Apple LLVM 5.1 - Clang 503.0.40)	For size (-Os)	iri_lt.o	1,362
ITK-LT-C	Windows 7 + GCC ARM Embedded for Windows	STM32F0DISCOVERY Board	32-bit	ARM Cortex-M0 (STM32F051)	GCC ARM Embedded 4.8.3	For size (-Os)	iri_lt.o	1,016
ITK-LT-C	Windows 7 + Code Composer Studio 6	MSP430FG4618/F201 Experimenter's Board	20-bit	TI MSP430 (MSP430FG4618)	TI Compiler 4.3.1	Level 4, for size	iri_lt.obj	1,508
ITK-LT-C	Windows 7 + AtmelStudio 6.1	AVR XMEGA-A1 Xplained	8-bit	Atmel AVR (ATxmega128A1)	AVR/GNU C Compiler 4.7.3	For size (-Os)	iri_lt.o	2,229

Source Code

Source code is provided for the ITK-LT-C library and example applications. This will enable users to port the IRI library as needed and to get a jump start on writing code for their own specific applications.

Notes on the Library

- New projects need to include all of the *.h and *.c files located within Library sub-directory
- As an exception to the rule above, only one platform_*.c need be included
- A PORTING.txt file has been provided with brief porting instructions
- A platform_empty.c file has been provided with a stubbed out API to quickly enable a porting effort

Library Source Code Contents

ITK_LT_C\Library\config.h

ITK_LT_C\Library\iri_lt.c

ITK_LT_C\Library\iri_lt.h

ITK_LT_C\Library\platform.h

ITK_LT_C\Library\platform_empty.c

ITK_LT_C\Library\platform_linux.c
ITK_LT_C\Library\platform_osx.c
ITK_LT_C\Library\platform_win32.c
ITK_LT_C\Library\PORTING.txt
ITK_LT_C\Library\version.h

Notes on the Examples

- **Example applications (all IRI_LT_*.c files) are provided with build systems for the following environments:**
 - Makefile for GNU based systems (Linux, Cygwin, etc.)
- Each example contains a comment block near the top of the file explaining the examples purpose
- An ipj_util_lt file is provided that contains functions common across the examples

Examples Source Code Contents

ITK_LT_C\Examples\ipj_util_lt.c
ITK_LT_C\Examples\ipj_util_lt.h
ITK_LT_C\Examples\IRI_LT_Access.c
ITK_LT_C\Examples\IRI_LT_Empty.c
ITK_LT_C\Examples\IRI_LT_GPIO.c
ITK_LT_C\Examples\IRI_LT_Intro.c
ITK_LT_C\Examples\IRI_LT_Loader.c
ITK_LT_C\Examples\IRI_LT_Multiple_Readers.c
ITK_LT_C\Examples\IRI_LT_Power_Management.c
ITK_LT_C\Examples\IRI_LT_RxOnly.c
ITK_LT_C\Examples\IRI_LT_Select.c
ITK_LT_C\Examples\IRI_LT_Test_CW_PRBS.c
ITK_LT_C\Examples\Makefile

API

The lists below provide links to all ITK-C API calls that *are* supported in ITK-LT-C.

Device Management:

- *ipj_get_api_version*
- *ipj_initialize_iri_device*
- *ipj_connect*
- *ipj_disconnect*

Operational Commands:

- *ipj_reset*
- *ipj_set*
- *ipj_set_value*
- *ipj_get*
- *ipj_get_value*
- *ipj_start*
- *ipj_stop*
- *ipj_receive*
- *ipj_flash_handle_loader_block*

Notes on *ipj_flash_handle_loader_block*

This API is disabled by default in order to keep the ITK-LT-C library size to a minimum.

It can be enabled via an `ENABLE_FW_UPDATES` compile time switch in `config.h`, at the cost of an increased library size.

See the `IRI_LT_Loader` example program for details on how to use this API to update firmware on the device.

Limitations

The lists below provide links to all ITK-C API calls that *are not* supported in ITK-LT-C.

Device Management:

- *ipj_deinitialize_iri_device*
- *ipj_register_handler*
- *ipj_modify_connection*
- *ipj_suppress_set_responses*
- *ipj_resume_set_responses*
- *ipj_set_receive_timeout_ms*

Operational Commands:

- *ipj_bulk_set*
- *ipj_bulk_get*
- *ipj_get_info*
- *ipj_resume*

In addition to API limitations, there is also no individual Response or Report timestamping. The Tag Operation Report however still has a configurable timestamp field.

Notes on Limitations

The sections below provide further explanations on a few specific API limitations.

ipj_register_handler The `Platform Interface` remains identical to ITK-C, however without the dynamic handler registration provided by `ipj_register_handler` the platform function names must be exactly as defined in `Platform.h`.

This is already the case for the provided `platform_*.c` files, so if they are used then no modification is needed.

Note: In addition to platform handlers, there is also an externed function defined in `iri_lt.h` for `ipj_report_handler` which needs to be implemented in user code. This allows users to trim down the report handler to only what is needed for a specific application. An example of `ipj_report_handler` can be found in `ipj_util_lt.c`.

ipj_modify_connection This function enabled dynamically changing the baudrate of the device on the fly. That is not possible with ITK-LT-C, however *Stored Settings* and the Indy Demo Tool can be used to configure a non-default baudrate which can then be passed into `ipj_connect`. In this way the ITK-LT-C can support alternate baudrates.

ipj_suppress_set_responses This function enabled dynamically suppressing set responses. This is not possible with ITK-LT-C, however a compile time switch has been provided in `config.h` to `SUPPRESS_SET_RESPONSES`. If this switch is enabled then any `Set` command will not send a response, thereby cutting down on the IRI traffic over the wire.

The risk with this option is to miss some possible errors that could occur with the command. Therefore it is recommended that `SUPPRESS_SET_RESPONSES` be disabled during development and only enabled when the host application is ready for production.

Reports

Reports are defined differently in ITK-LT-C. All of them are simple c structs with each parameter defined as a fixed width `uint32_t`.

The exception to this rule is the Tag Operation Report, which is encoded as a series of simple field blocks separated by a 32-bit header (16-bits `FIELD_ID` and 16-bits `FIELD_LENGTH`). This allows users to trim down the report handler to only what is needed for a specific application. The desired report fields can be enabled or disabled through *E_IPJ_KEY_REPORT_CONTROL_TAG*.

Source code for both report types is provided below.

Definitions

Here is the definition of the report structs and the Tag Operation Report field IDs from `iri_lt.h`:

```

// *****
typedef struct _ipj_stop_report
{
    uint32_t error;
    uint32_t action;
} ipj_stop_report;

typedef struct _ipj_gpio_report
{
    uint32_t gpio_modes[5];
    uint32_t gpio_states[5];
} ipj_gpio_report;

typedef struct _ipj_error_report
{
    uint32_t error;
    uint32_t param1;
    uint32_t param2;
    uint32_t param3;
    uint32_t param4;
} ipj_error_report;

typedef struct _ipj_status_report
{
    uint32_t status_flag;
    uint32_t status_1;
    uint32_t status_2;
    uint32_t status_3;
    uint32_t data[16];
} ipj_status_report;

typedef struct _ipj_test_report
{
    uint32_t error;
    uint32_t test_id;
    uint32_t result_1;
    uint32_t result_2;
    uint32_t result_3;
    uint32_t data[16];
} ipj_test_report;

// *****
// IRI API - LT Defines
// *****
#define E_IPJ_TAG_OPERATION_REPORT_ERROR 100
#define E_IPJ_TAG_OPERATION_REPORT_EPC 201
#define E_IPJ_TAG_OPERATION_REPORT_TID 202
#define E_IPJ_TAG_OPERATION_REPORT_PC 203
#define E_IPJ_TAG_OPERATION_REPORT_XPC 204
#define E_IPJ_TAG_OPERATION_REPORT_CRC 205
#define E_IPJ_TAG_OPERATION_REPORT_TIMESTAMP 206
#define E_IPJ_TAG_OPERATION_REPORT_RSSI 207
#define E_IPJ_TAG_OPERATION_REPORT_PHASE 208
#define E_IPJ_TAG_OPERATION_REPORT_CHANNEL 209
#define E_IPJ_TAG_OPERATION_REPORT_ANTENNA 210
#define E_IPJ_TAG_OPERATION_REPORT_TAG_OPERATION_TYPE 300
#define E_IPJ_TAG_OPERATION_REPORT_TAG_OPERATION_DATA 400
#define E_IPJ_TAG_OPERATION_REPORT_RETRIES 500

```

```
#define E_IPJ_TAG_OPERATION_REPORT_DIAGNOSTIC 600
```

Here are some macros for helping decode the Tag Operation Report:

```
/* Report Decoding */
#define REPORT_FIELD_ID(x) ((x >> 16) & 0xFFFF)
#define REPORT_FIELD_LENGTH(x) (x & 0xFFFF)
#define ROUNDED_COUNT_32(x) (((x + sizeof(uint32_t) - 1) & 0xFFFFFFFF) / sizeof(uint32_t))
```

Decoders

Here are example report decoders, including the Tag Operation Report, as defined in ipj_util_lt.c:

```
ipj_error ipj_util_tag_operation_report_handler(
    ipj_iri_device* iri_device,
    uint32_t report_count_32,
    uint32_t* tag_operation_report)
{
    uint32_t index = 0;
    while (index < report_count_32)
    {
        uint32_t field_header = tag_operation_report[index++];
        switch (REPORT_FIELD_ID(field_header))
        {
            case E_IPJ_TAG_OPERATION_REPORT_ERROR:
            {
                IPJ_UTIL_PRINT_ERROR(tag_operation_report[index], "tag_operation_report");
                break;
            }
            case E_IPJ_TAG_OPERATION_REPORT_PC:
            {
                printf("PC: %04X\n", tag_operation_report[index]);
                break;
            }
            case E_IPJ_TAG_OPERATION_REPORT_XPC:
            {
                printf("XPC: %04X\n", tag_operation_report[index]);
                break;
            }
            case E_IPJ_TAG_OPERATION_REPORT_CRC:
            {
                printf("CRC: %04X\n", tag_operation_report[index]);
                break;
            }
            case E_IPJ_TAG_OPERATION_REPORT_RSSI:
            {
                printf("RSSI: %d\n", (int)tag_operation_report[index]);
                break;
            }
            case E_IPJ_TAG_OPERATION_REPORT_PHASE:
            {
                printf("PHASE: %d\n", (int)tag_operation_report[index]);
                break;
            }
            case E_IPJ_TAG_OPERATION_REPORT_CHANNEL:
            {
                printf("CHANNEL: %d\n", tag_operation_report[index]);
            }
        }
    }
}
```

```

        break;
    }
    case E_IPJ_TAG_OPERATION_REPORT_ANTENNA:
    {
        printf("ANTENNA: %d\n", tag_operation_report[index]);
        break;
    }
    case E_IPJ_TAG_OPERATION_REPORT_TAG_OPERATION_TYPE:
    {
        printf("Tag operation: ");
        switch (tag_operation_report[index])
        {
            case E_IPJ_TAG_OPERATION_TYPE_READ:
            {
                printf("READ\n");
                break;
            }
            case E_IPJ_TAG_OPERATION_TYPE_WRITE:
            {
                printf("WRITE\n");
                break;
            }
            case E_IPJ_TAG_OPERATION_TYPE_LOCK:
            {
                printf("LOCK\n");
                break;
            }
            case E_IPJ_TAG_OPERATION_TYPE_KILL:
            {
                printf("KILL\n");
                break;
            }
            case E_IPJ_TAG_OPERATION_TYPE_BLOCKPERMALOCK:
            {
                printf("BLOCK PERMALOCK\n");
                break;
            }
            case E_IPJ_TAG_OPERATION_TYPE_WRITE_EPC:
            {
                printf("WRITE EPC\n");
                break;
            }
            case E_IPJ_TAG_OPERATION_TYPE_QT:
            {
                printf("QT\n");
                break;
            }
            case E_IPJ_TAG_OPERATION_TYPE_CUSTOM:
            {
                printf("CUSTOM\n");
                break;
            }
        }
        break;
    }
    case E_IPJ_TAG_OPERATION_REPORT_RETRIES:
    {
        printf("RETRIES: %d\n", tag_operation_report[index]);
    }

```

```

        break;
    }
    case E_IPJ_TAG_OPERATION_REPORT_DIAGNOSTIC:
    {
        printf("DIAGNOSTIC: %X\n", tag_operation_report[index]);
        break;
    }
    case E_IPJ_TAG_OPERATION_REPORT_TIMESTAMP:
    {
        printf(
            "TIMESTAMP: %lld\n",
            (uint64_t) tag_operation_report[index]
            | ((uint64_t) (tag_operation_report[index + 1]) << 32));
        break;
    }
    case E_IPJ_TAG_OPERATION_REPORT_EPC:
    {
        printf("EPC: ");
        ipj_util_print_epc(
            (uint16_t*) &tag_operation_report[index],
            REPORT_FIELD_LENGTH(field_header) / sizeof(uint16_t),
            true);
        break;
    }
    case E_IPJ_TAG_OPERATION_REPORT_TID:
    {
        printf("TID: ");
        /* Reuse the print epc method */
        ipj_util_print_epc(
            (uint16_t*) &tag_operation_report[index],
            REPORT_FIELD_LENGTH(field_header) / sizeof(uint16_t),
            true);
        break;
    }
    case E_IPJ_TAG_OPERATION_REPORT_TAG_OPERATION_DATA:
    {
        printf("Report contains data: %d bytes\n", REPORT_FIELD_LENGTH(field_header));
        /* Reuse the print epc method */
        ipj_util_print_epc(
            (uint16_t*) &tag_operation_report[index],
            REPORT_FIELD_LENGTH(field_header) / sizeof(uint16_t),
            true);
        break;
    }
}

index += ROUNDED_COUNT_32(REPORT_FIELD_LENGTH(field_header));

}

ipj_util_print_divider('-', 80);
return E_IPJ_ERROR_SUCCESS;
}

/* Stop report handler processes asynchronous reports */
ipj_error ipj_util_stop_report_handler(
    ipj_iri_device* iri_device,
    ipj_stop_report* ipj_stop_report)

```

```

{
    if (ipj_stop_report->error == E_IPJ_ERROR_SUCCESS)
    {
        /* Print reader identifier */
        printf("%s: STOPPED\n", (char*) iri_device->reader_identifier);

        /* Set the stopped flag, the stop report does not have any fields that
         * need to be checked */
        ipj_stopped_flag = 1;
    }
    else
    {
        IPJ_UTIL_PRINT_ERROR(ipj_stop_report->error, "stop_report");
    }
    return ipj_stop_report->error;
}

/* GPIO handler processes asynchronous GPIO event reports */
ipj_error ipj_util_gpio_report_handler(
    ipj_iri_device* iri_device,
    ipj_gpio_report* gpio_report)
{
    unsigned int i;
    for (i = 0; i < 40; i++)
        printf("*");

    printf("\n%s: GPIO Report\n", (char*) iri_device->reader_identifier);

    printf("GPIO Modes: ");
    for (i = 1; i < ARRAY_SIZE(gpio_report->gpio_modes); i++)
    {
        switch (gpio_report->gpio_modes[i])
        {
            case E_IPJ_GPIO_MODE_DISABLED:
            {
                printf("D ");
                break;
            }
            case E_IPJ_GPIO_MODE_INPUT:
            {
                printf("I ");
                break;
            }
            case E_IPJ_GPIO_MODE_OUTPUT:
            {
                printf("O ");
                break;
            }
            case E_IPJ_GPIO_MODE_OUTPUT_PULSE:
            {
                printf("OP ");
                break;
            }
            case E_IPJ_GPIO_MODE_INPUT_ACTION:
            {
                printf("IA ");
                break;
            }
        }
    }
}

```

```

        case E_IPJ_GPIO_MODE_OUTPUT_ACTION:
        {
            printf("OA ");
            break;
        }
        case E_IPJ_GPIO_MODE_OUTPUT_PULSE_ACTION:
        {
            printf("OPA");
            break;
        }
    }

    if (i < ARRAY_SIZE(gpio_report->gpio_modes) - 1)
        printf("|");
}
printf("\nGPIO States: ");

for (i = 1; i < ARRAY_SIZE(gpio_report->gpio_states); i++)
{
    switch (gpio_report->gpio_states[i])
    {
        case E_IPJ_GPIO_STATE_LO:
        case E_IPJ_GPIO_STATE_FLOAT:
        {
            printf("0 ");
            break;
        }
        case E_IPJ_GPIO_STATE_HI:
        {
            printf("1 ");
            break;
        }
    }

    if (i < ARRAY_SIZE(gpio_report->gpio_states) - 1)
        printf("|");
}
printf("\n");

for (i = 0; i < 40; i++)
    printf("*");

printf("\n");

return E_IPJ_ERROR_SUCCESS;
}

ipj_error ipj_util_error_report_handler(
    ipj_iri_device* iri_device,
    ipj_error_report* error_report)
{
    printf("Error Report\n");
    printf("Error: 0x%X, %d\n", error_report->error, error_report->error);
    printf("Param1: 0x%X, %d\n", error_report->param1, error_report->param1);
    printf("Param2: 0x%X, %d\n", error_report->param2, error_report->param2);
    printf("Param3: 0x%X, %d\n", error_report->param3, error_report->param3);
    printf("Param4: 0x%X, %d\n", error_report->param4, error_report->param4);
    return E_IPJ_ERROR_SUCCESS;
}

```

```

}

ipj_error ipj_util_status_report_handler(
    ipj_iri_device* iri_device,
    ipj_status_report* status_report)
{
    uint32_t i;
    printf("Status Report\n");
    printf("status_flag: 0x%X, %d\n", status_report->status_flag, status_report->status_flag);
    printf("status_1: 0x%X, %d\n", status_report->status_1, status_report->status_1);
    printf("status_2: 0x%X, %d\n", status_report->status_2, status_report->status_2);
    printf("status_3: 0x%X, %d\n", status_report->status_3, status_report->status_3);

    printf("Additional Data:\n");
    for (i = 0; i < ARRAY_SIZE(status_report->data); i++)
    {
        printf("[%d] 0x%X, %d\n", i, status_report->data[i], status_report->data[i]);
    }

    return E_IPJ_ERROR_SUCCESS;
}

ipj_error ipj_util_test_report_handler(
    ipj_iri_device* iri_device,
    ipj_test_report* test_report)
{
    uint32_t i;
    if (test_report->error != E_IPJ_ERROR_SUCCESS)
    {
        IPJ_UTIL_PRINT_ERROR(test_report->error, "test");
    }

    printf("\n*** %s Test Report ***\n", (char*) iri_device->reader_identifier);
    printf("Test ID: %d\n", test_report->test_id);
    printf("Test Results (optional):\n");
    printf("Result 1: 0x%X, %d\n", test_report->result_1, test_report->result_1);
    printf("Result 2: 0x%X, %d\n", test_report->result_2, test_report->result_2);
    printf("Result 3: 0x%X, %d\n", test_report->result_3, test_report->result_3);
    printf("Additional Data:\n");

    for (i = 0; i < ARRAY_SIZE(test_report->data); i++)
    {
        printf("[%d] 0x%X, %d\n", i, test_report->data[i], test_report->data[i]);
    }

    return E_IPJ_ERROR_SUCCESS;
}

```

1.5.4 Frequently Asked Questions (FAQ)

Introduction

The document provides general frequently asked questions and answers about the Indy ITK Release and the Indy SiPs. For more information about the Indy SiPs, see their Datasheets and Hardware User's Guides, which are included in the Release package under `.\Documentation\`, or on the web [here \(for RS500\)](#).

General IRI and Indy SiP Questions

What are the differences between ITK-C and ITK-LT-C?

The primary differences between ITK-C and ITK-LT-C are that the ITK-C implements more of the Indy SiP's functionality, at the cost of more program memory (ROM/FLASH) and RAM consumption. The ITK-LT-C may allow implementation of the Indy SiP interface on microcontrollers and microprocessors with memory. ITK-C's library implements the full capabilities of the Indy SiP's built in firmware, including operations like bulk set and get, and bootloading. ITK-LT-C's library only implements a limited subset of Indy SiP's built in firmware. ITK-C consumes between 9 and 16 kB of ROM, and ITK-LT-C consumes between 1 and 3 kB of ROM. For more details on the code consumption of each of the libraries, see *ITK-C Memory Usage* and *ITK-LT-C Memory Usage*. For more details on the limitations of ITK-LT-C, see the *Limitations* section of the ITK-LT-C API guide.

ITK-LT-C should only be used if the the standard ITK-C can not be used due to host memory constraints. The ITK-C is extensible, has broader capabilities, and enables future advanced features.

What is the persistence of setting a key value?

Setting a key value will persist until the value is changed again or the device is reset.

Caution should be used when setting one of the various ENABLE keys. This is particularly important when using keys which control the RFID protocol enabling (eg. *Select* and *Tag Access*). If one of these modes are enabled (eg. via *E_IPJ_KEY_SELECT_ENABLE* or *E_IPJ_KEY_TAG_OPERATION_ENABLE*), they must be set to disabled if they are not desired on next command.

Stored settings can be used to configure the power on default values of writeable key. So for persistent values across reset, please see *Stored Settings*.

Do the Indy SiPs have a Recovery Mode?

When installing the image updates for the Application for the first time or when attempting to recover a Indy SiP, it is sometimes necessary to force the Indy SiP into recovery mode manually. This can be done by jumpering the WKUP pin high and resetting the board.

- Using a jumper, connect the WKUP pin to 3.3V (see jumper J4 on the development kit)
- Press the Reset Button once
- The board should now be in recovery mode with the Health and Status LEDs blinking
- You can now attempt to re-flash the application image
- Once the image has been successfully flashed, remove the jumper, the device should automatically jump to the new application

What is the behavior of the HEALTH pin?

The HEALTH pin indicates whether the Indy SiP is operating in its normal mode, or if some other condition exists. The pin is cycled high and low in specific patterns to indicate the state of the Indy SiP. Those patterns are as follows:

Boot:

- High

Idle:

- 1 second high, 1 second low

Active:

- 250ms high, 750 second low

Watchdog reset has occurred:

- Low

Recovery mode:

- Alternate pattern with STATUS pin

What is the behaviour of the STATUS pin?

The STATUS pin indicates whether the Indy SiP is operating in its active mode, or if some other condition exists. The pin is cycled high and low in specific patterns to indicate the state of the Indy SiP. Those patterns are as follows:

Boot:

- Toggle Once

Idle:

- Off

Active:

- During inventory, the high time is between 150ms and 750ms based on the number of tags in the field. The high time is 1000ms minus the high time. If there are no tags in the field the pin remains low.

Watchdog reset has occurred:

- Alternate high and low

Recovery mode:

- Alternate pattern with HEALTH pin

What power management modes are available?

The software power management modes available are standard idle, low latency idle, standby and sleep. For more information please see the *Power Management* example program.

The RS2000 SiP also has a “shutdown” mode, which is a hardware power mode that can be entered by applying 0 V to the Enable pin. In this mode, the regulators in the RS2000 SiP are disabled.

How can the Indy SiPs be woken up from Sleep Mode?

Set WKUP pin high to wake up the Indy SiP. Set WKUP low to continue with normal operation. Customers can leave WKUP disconnected if they do not intend on using sleep mode. The WKUP pin can also be used to wake from from standby.

Can the baud rate be changed to 1200?

Yes the Baud Rate can be changed. Please see the *IRI_Change_Baudrate Example*.

What does the RSSI field in the Tag Report mean?

The reader measures the backscatter power from the tag. The measured power is referenced to the reader RF port. The units are in centi-dB-mW (cdBm), and the value is signed.

What are the default RF Profile operating parameters?

The default RF Parameters are described in the following table.

Table 1.2: RF Parameters

Parameter	Value
Tari	25.0 us
RTCAl	62.5 us
TRCal	85.3 us
Tari	25.0 us
Divide Ratio	64/3
M	Miller 4
BLF	250 kHz
Tpri	4 us
BLF	250 kHz

Note: RF Profile subject to change

Please see *RF Mode* for all available modes.

What are the host processor recommendations?

When selecting a host processor, we recommend the following set of guidelines:

- 32 bit microcontroller running at at least 8MHz
- 32k bytes of NVM
- 10k bytes RAM
- A USART capable of 115200 baud
- Ability to address 8 bit bytes i.e.: `sizeof(uint8_t) == 1`, `sizeof(uint32_t) == 4`. (Other word/byte widths are possible, but not fully supported).

Please note that these are not *minimums*, but will ensure your ability to exercise 100% of the Indy SiPs' capabilities.

Please contact us at support.impinj.com for assistance if you have a host processor with capabilities lower than the recommended guidelines.

How can I uniquely identify each individual Indy SiP device?

The identifier printed on the label of the device is a combination of the SKU, Lot Date Code, and Serial Number from a lot.

SERIAL #: XXZZWWYYAAAA

- XX is the SKU (1 = GX, 2 = EU)
- ZZ is the Lot Number

- WW is the Work Week produced
- YY is the Year Produced
- AAAA is the Serial number within the lot

Each individual value can be retrieved via the following keys:

- *E_IPJ_KEY_PRODUCT_SKU*
- *E_IPJ_KEY_LOT_DATE_CODE*
- *E_IPJ_KEY_UNIQUE_ID*

Alternatively, the combined unique identifier can be retrieved via the following key (which has 2 32-bit values):

- *E_IPJ_KEY_UNIQUE_ID*

For example, with a SERIAL #: 010151130247

- SKU = 1
- LOT_DATE_CODE = 15113
- SERIAL_NUMBER = 247
- UNIQUE_ID[0] = 0x5D0DF487
- UNIQUE_ID[1] = 0x00000002

Which version of Linux has the ITK-C been tested with?

The ITK-C is compatible with a wide variety of linux distributions. It has been tested with Ubuntu 14.04 and Centos 6. Any distribution with support for the termios serial port API should be compatible.

RS500 Specific Questions

What is the typical current consumption for the RS500?

The table below is provided for reference. For detailed electrical specs, see the Indy RS500 Datasheet, which is included in the RS500 Release package under `.\Documentation\`, or on the web [here](#).

Table 1.3: Current Consumption

Mode	Current (+/- 10%)
Active	510 mA (GX), 580 mA (EU)
Low Latency Idle	50 mA
Standard Idle	15 mA
Standby	1 mA
Sleep	100 uA

How should the pins of the RS500 be connected?

The details below are provided for reference. For more details on RS500 pin recommendations, see the Indy RS500 Hardware User's Guide, which is included in the Release package under `.\Documentation\`, or on the web [here](#) (for RS500).

Required connections:

- VDC_IN and GND are required to power the RS500.
- RF is required to connect to the UHF RFID antenna.
- UART1 Tx and Rx are required to communicate with the system host.

Recommended connections:

- nRST is used to reset the RS500 if UART communication is not available. This connection is highly recommended. This pin is sometimes internally driven strong low, so it should be driven by an open drain signal. It must not be driven strong high.
- UART2 Tx and Rx may be used to examine debug information.
- HEALTH indicates successful operation of the RS500. Connection to an LED provides a visual indication of whether or not an error condition exists.
- STATUS provides an indication when the RS500 is in active mode (for example, inventorying tags). Connection to an LED provides a visual indicator of the device's activity.

Optional connections:

- GPIOs allow interaction with the RS500 as both digital inputs and outputs. They may be used to trigger inventory, generate events based on inventory activity, or provide general-purpose user-controlled digital I/O.
- WKUP provides a mechanism to wake up the RS500 from the low power Sleep mode on a rising edge. The pin must go low for the device to fully wakeup. WKUP is also used to force entry into the Impinj firmware bootstrap. If unused, this pin should be tied to logic low.
- UC_ADC allows use of an ADC to convert an analog input voltage into a digital value.
- UC_DAC allows use of a DAC to generate an analog output voltage from a digital value.
- BOOT0 provides access to the built-in bootloader in case the Impinj firmware bootstrap is corrupted. For more details on the built-in bootloader, please contact Impinj support.

No connect:

- SWCLK and SWD connections are reserved for Impinj use only.

What is the drive strength of the RS500 GPIO pins?

The GPIO pins can sink and source 8 mA at 3.3V.

For detailed electrical specs and GPIO behaviour, see the Indy RS500 Datasheet and Indy RS500 Hardware User's Guide, which are included in the Release package under `.\Documentation\`, or on the web [here \(for RS500\)](#).

How can RS500 devices with firmware older than v0.8.1.0 be upgraded?

If an RS500 has firmware from a release older than v0.8.1.0, it cannot be upgraded using this release. To get help updating an RS500 with firmware from a release older than v0.8.1.0, please submit a support ticket at support.impinj.com.

What are the typical latency times to enter the Idle state for the RS500?

Table 1.4: Time To Idle Latency

Mode	Latency Time
Hard Reset	200 ms
Soft Reset	200 ms
Standby	50 ms
Sleep	200 ms

How long must nRST be held low to reset the RS500?

The nRST pin must be held low for a minimum of 25 us to reset the device.

What are the inventory rates of the RS500?

The following are estimates of typical RS500 inventory rates given certain population estimates and tags present. Actual inventory rates will vary depending on a number of factors.

Table 1.5: Inventory Rates

Tags in Field	Initial Population Estimate	Inventory Rate (tags/sec)
1	1	130
1	16	35
16	16	50

What is the behavior of the RS500 NRST pin?

Customers must not apply a strong high voltage source to the NRST pin. The RS500 must be able to pull NRST low in order to reset itself. Customers can connect an open-drain source and drive the pin strong low in order to reset the RS500. Customers can leave the NRST pin disconnected if they do not need to reset the RS500.

RS2000 Specific Questions**How are the antennas of the RS2000 switched?**

RS2000's antenna switching is configured using the *E_IPJ_KEY_ANTENNA_SEQUENCE* key.

For more details, see the configuration example *Antenna Switching*.

How is RS2000's temperature monitored and controlled?

RS2000, with its 31.5 dBm maximum power output, can easily self-heat above its maximum operating temperature under certain thermal conditions. It may be necessary to monitor and control operating parameters to ensure reliable operation.

There is an example on how to read RS2000's internal temperatures in the configuration example *Get Temperature*.

RS2000's firmware monitors the power amplifier (PA) temperature, and automatically stops RFID operations if it exceeds 85 degrees Celsius. When this occurs, a stop report is sent with error code

`E_IPJ_ERROR_LIMIT_PA_TEMPERATURE_MAX` which indicates that the PA has exceeded this temperature. Start commands may be sent again immediately, but may be stopped automatically by subsequent over-temperature conditions. It is advisable to monitor for over-temperature events, and add some activity hold-off or decrease in self-heating via transmit power reduction.

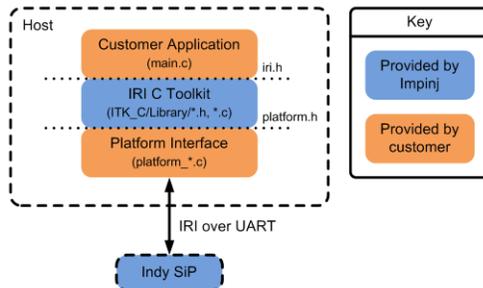
Future versions of RS2000 firmware will include an on-board control system that will automatically keep the device operating at decreased activity or power levels.

1.5.5 Overview

Architecture

Customer applications communicate with the Indy SiPs using the Impinj Radio Interface (IRI) API. Details of the IRI API are defined in the `iri.h` header file and includes *functions*, *structures*, *defines*, and *report handlers*.

The IRI library also exposes a *platform interface*, see `platform.h`. The platform interface provides functions to open and close the serial port, read and write the serial port. Platform interface also provides functions to generate timestamps and sleep. Source code examples for the platform implementation are provided for Windows, Linux and OSX. Customers can use the example source code or write their own to port to a new platform.



IRI Communication Model

At power-up, or following a reset, the Host communicates synchronously with the Indy SiP. The Host sends a command to the Indy SiP and the Indy SiP sends a response to the Host. All of the IRI API functions that require communication with the Indy SiP follow this standard, synchronous protocol (`ipj_reset`, `ipj_set`, `ipj_set_value`, `ipj_bulk_set`, `ipj_get`, `ipj_get_value`, `ipj_bulk_get`, `ipj_start`, and `ipj_stop`).

After the IRI host calls `ipj_start`, the Indy SiP transitions to an asynchronous communication protocol. The Indy SiP generates reports when certain events occur (e.g. Tag Operation Reports when Tags are Inventoried, Stop Reports when Inventory stops, GPIO Reports when GPIO status changes, or Error Reports when errors are detected). The IRI host is responsible for calling `ipj_receive` to retrieve the reports generated by the Indy SiP. The IRI host is also responsible for providing a report handler to process the reports. The Indy SiP will issue a Stop Report before transitioning back to the standard, synchronous protocol. The Indy SiP may generate reports after a stop command has been issued and the corresponding response received. The IRI host is responsible for calling `ipj_receive` until the Stop Report is received.

Typical Host Application Elements

`IRI_Intro` is a simple Host application. `IRI_Intro` uses the elements of a typical Host application shown below:

1. Allocate or declare IRI device
2. Initialize IRI device structure
3. Connect to IRI device

4. Register platform handlers
 - Platform open handler
 - Platform close handler
 - Platform transmit handler
 - Platform receive handler
 - Platform timestamp handler
 - Platform sleep handler
5. Register report handler
6. Configure the Indy SiP
 - See *IRI Configuration Examples* for more details
7. Start inventory
8. While running inventory
 - Call *ipj_receive* to process incoming reports
9. Stop inventory
 - Call *ipj_receive* to process incoming reports
 - Stop when stop report received, indicating that the Indy SiP has stopped inventory
10. Disconnect from IRI device
11. De-initialize IRI device structure

IRI API functions that transmit commands to the Indy SiP wait for a response from the Indy SiP before returning (e.g. *ipj_reset*, *ipj_set*, *ipj_set_value*, *ipj_bulk_set*, *ipj_get*, *ipj_get_value*, *ipj_bulk_get*, *ipj_start*, and *ipj_stop*).

While Inventory is running the Indy SiP generates tag operation reports for each tag that is inventoried. The Host application calls *ipj_receive* to process the incoming tag operation reports. The *ipj_receive* function processes the data available from the serial driver, calls the report handler upon detecting a complete report, and returns. The *ipj_receive* function does not wait for incoming report data. The Host application provides a report handler function to process the incoming tag operation reports.

ITK-C Directory Structure

Source code is provided for the IRI library and example applications to enable users to port the IRI library.

Library

- New projects need to include all of the `*.h` and `*.c` files located within `ITK_C/Library` directory
- As an exception to the rule above, only one `platform_*.c` need be included
- A `PORTING.txt` file has been provided with brief porting instructions
- A `platform_empty.c` file has been provided with a stubbed out API to quickly enable a porting effort

IRI library source code contents:

ITK_C\Library\config.h
ITK_C\Library\iri.c
ITK_C\Library\iri.h
ITK_C\Library\platform.h
ITK_C\Library\platform_empty.c
ITK_C\Library\platform_linux.c
ITK_C\Library\platform_osx.c
ITK_C\Library\platform_win32.c
ITK_C\Library\PORTING.txt
ITK_C\Library\version.h
ITK_C\Library\Nanopb\pb.h
ITK_C\Library\Nanopb\pb_decode.c
ITK_C\Library\Nanopb\pb_decode.h
ITK_C\Library\Nanopb\pb_encode.c
ITK_C\Library\Nanopb\pb_encode.h
ITK_C\Library\PbMessages\commands.pb.c
ITK_C\Library\PbMessages\commands.pb.h
ITK_C\Library\PbMessages\enums.pb.h
ITK_C\Library\PbMessages\error_codes.pb.h
ITK_C\Library\PbMessages\key_codes.pb.h
ITK_C\Library\PbMessages\messages.pb.c
ITK_C\Library\PbMessages\messages.pb.h
ITK_C\Library\PbMessages\packet.pb.c
ITK_C\Library\PbMessages\packet.pb.h

Examples

- **Example applications (all IRI_*.c files) are provided with build systems for the following environments:**
 - Makefile for GNU based systems (Linux, Cygwin, etc.)
 - Visual Studio solutions for MSVC 2005 and MSVC 2012.
- **The Visual Studio solutions contain:**

- A `README.txt` containing instructions on how to create a new IRI example project
- An *IRI_Empty* project that is already set up as a placeholder to quickly try out customized IRI commands.
- An *ipj_util* project is provided that contains functions common across the examples

Examples source code contents:

```
ITK_C\Examples\ipj_util.c
ITK_C\Examples\ipj_util.h
ITK_C\Examples\IRI_Access.c
ITK_C\Examples\IRI_Change_Baudrate.c
ITK_C\Examples\IRI_Empty.c
ITK_C\Examples\IRI_GPIO.c
ITK_C\Examples\IRI_Intro.c
ITK_C\Examples\IRI_Loader.c
ITK_C\Examples\IRI_Multiple_Readers.c
ITK_C\Examples\IRI_Power_Management.c
ITK_C\Examples\IRI_Select.c
ITK_C\Examples\IRI_Test_CW_PRBS.c
ITK_C\Examples\Makefile
ITK_C\Examples\VS2005\Examples.sln
ITK_C\Examples\VS2005\README.txt
ITK_C\Examples\VS2005\stdbool.h
ITK_C\Examples\VS2005\stdint.h
ITK_C\Examples\VS2012\Examples.sln
ITK_C\Examples\VS2012\README.txt
ITK_C\Examples\VS2012\stdbool.h
```

1.5.6 IRI Example Programs

The ITK includes a number of example programs for Windows, Linux, and OS X. These examples demonstrate many of the features of the ITK and Indy Reader SiPs.

Examples Directory Structure

Several example IRI programs (all `IRI_*.c` files) are provided with the release.

```
ITK_C\Examples\ipj_util.c
ITK_C\Examples\ipj_util.h
ITK_C\Examples\IRI_Access.c
ITK_C\Examples\IRI_Change_Baudrate.c
ITK_C\Examples\IRI_Empty.c
ITK_C\Examples\IRI_GPIO.c
ITK_C\Examples\IRI_Intro.c
ITK_C\Examples\IRI_Loader.c
ITK_C\Examples\IRI_Multiple_Readers.c
ITK_C\Examples\IRI_Power_Management.c
ITK_C\Examples\IRI_Select.c
ITK_C\Examples\IRI_Test_CW_PRBS.c
ITK_C\Examples\Makefile
ITK_C\Examples\VS2005\Examples.sln
ITK_C\Examples\VS2005\README.txt
ITK_C\Examples\VS2005\stdbool.h
ITK_C\Examples\VS2005\stdint.h
ITK_C\Examples\VS2012\Examples.sln
ITK_C\Examples\VS2012\README.txt
ITK_C\Examples\VS2012\stdbool.h
```

Building, Running, and Debugging the Examples

To build the examples, open the one of the Examples.sln Visual Studio solutions or type *make* in the Examples directory.

To run the examples in Windows in the command line, run the executable using the correct COM port as the argument, for example, type `IRI_Intro COM1`.

To run the examples in Linux in the console, run the output of the build using the `/dev/` device as the argument, for example, type `output/IRI_Intro /dev/ttyUSB0`.

To debug the examples in Visual Studio, perform the following steps:

1. Right click on the desired example and select “Set as Startup Project”.
2. Open the properties of the desired example project by right clicking the project and selecting “Properties”.
3. Under “Configuration Properties”, select “Debugging”.

4. In the “Command Arguments” field, enter the COM port that connects to your Indy device (e.g. COM1).
5. Under “Configuration Properties”, expand “Linker” and select “Debugging”.
6. Set the “Generate Debug Info” setting to “Yes (/DEBUG)”.
7. Close the project configuration dialog.
8. Press the “Local Windows Debugger” button, and debugging should begin.

IRI_Intro Example

The IRI_Intro example connects to an Indy SiP and performs basic inventory for 1 second, printing read EPCs to the console or command line.

IRI_Intro - Source Code

IRI_Intro source code is provided in IRI_Intro.c :

```

/*
*****
*
*          IMPINJ CONFIDENTIAL AND PROPRIETARY
*
* This source code is the sole property of Impinj, Inc. Reproduction or
* utilization of this source code in whole or in part is forbidden without
* the prior written consent of Impinj, Inc.
*
* (c) Copyright Impinj, Inc. 2013-2015. All rights reserved.
*
*****/
#include <stdio.h>
#include <string.h>
#include "ipj_util.h"
#include "iri.h"

/* PURPOSE: This example illustrates the use of the basic inventory operation
   and to retrieve RS500 information. */

/* Parameters */
#define IPJ_EXAMPLE_DURATION_MS 1000

/* Allocate memory for iri device */
static ipj_iri_device iri_device = { 0 };

/* Main */
int main(int argc, char* argv[])
{
    /* Define error code */
    ipj_error error = E_IPJ_ERROR_SUCCESS;
    uint32_t value;
    ipj_key_info keyinfo;

    IPJ_UTIL_CHECK_USER_INPUT_FOR_COM_PORT_RETURN_ON_ERROR()

    /* Common example setup */
    error = ipj_util_setup(&iri_device, argv[1]);
    IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_util_setup");

```

```

/* Reader Info */
printf("RS500 Info\n");
error = ipj_get_value(&iri_device, E_IPJ_KEY_SERIAL_NUMBER, &value);
IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_get_value E_IPJ_KEY_SERIAL_NUMBER");
printf("Serial Number      : %d\n", value);
ipj_get_value(&iri_device, E_IPJ_KEY_BOOTSTRAP_VERSION, &value);
printf("Bootstrap Version   : 0x%08X\n", value);
ipj_get_value(&iri_device, E_IPJ_KEY_BOOTSTRAP_CRC, &value);
printf("Bootstrap CRC       : 0x%08X\n", value);
ipj_get_value(&iri_device, E_IPJ_KEY_APPLICATION_VERSION, &value);
printf("Application Version: 0x%08X\n", value);
ipj_get_value(&iri_device, E_IPJ_KEY_APPLICATION_CRC, &value);
printf("Application CRC      : 0x%08X\n", value);
ipj_get_value(&iri_device, E_IPJ_KEY_MICROPROCESSOR_ID, &value);
printf("Microprocessor      : 0x%08X\n", value);
ipj_get(&iri_device, E_IPJ_KEY_MICROPROCESSOR_ID, 0, 1, &value);
printf("Microprocessor Id   : 0x%08X", value);
ipj_get(&iri_device, E_IPJ_KEY_MICROPROCESSOR_ID, 0, 2, &value);
printf("-0x%08X", value);
ipj_get(&iri_device, E_IPJ_KEY_MICROPROCESSOR_ID, 0, 3, &value);
printf("-0x%08X", value);
printf("\n\n");

IPJ_CLEAR_STRUCT(keyinfo);

/* Verify that we can indeed write the ANTENNA_TX_POWER key */
error = ipj_get_info(&iri_device, E_IPJ_KEY_ANTENNA_TX_POWER, &keyinfo);
IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_get_info E_IPJ_KEY_ANTENNA_TX_POWER");

/* Check to make sure the key has either write, or read/write permissions */
if (keyinfo.key_permissions == E_IPJ_KEY_PERMISSIONS_READ_ONLY)
{
    printf("ERROR: Unable to set ANTENNA_TX_POWER KEY\n\n");
    return -1;
}

/* Configure transmit power */
error = ipj_set_value(&iri_device, E_IPJ_KEY_ANTENNA_TX_POWER, 2300);
IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_set_value E_IPJ_KEY_ANTENNA_TX_POWER");

/* Start inventory */
error = ipj_util_perform_inventory(&iri_device, IPJ_EXAMPLE_DURATION_MS);
IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_util_perform_inventory");

/* Common example cleanup */
error = ipj_util_cleanup(&iri_device);
IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_util_cleanup");

return error;
}

```

IRI_Access Example

The IRI_Access example demonstrates how to use the Access tag operation to write a tag's EPC.

This example can be easily modified to write user memory by modifying the values of the `E_IPJ_KEY_WRITE_MEM_BANK` and `E_IPJ_KEY_WRITE_WORD_POINTER` keys.

IRI_Access - Source Code

IRI_Access source code is provided in IRI_Access.c :

```

/*
*****
*
*          IMPINJ CONFIDENTIAL AND PROPRIETARY
*
*
* This source code is the sole property of Impinj, Inc. Reproduction or
* utilization of this source code in whole or in part is forbidden without
* the prior written consent of Impinj, Inc.
*
*
* (c) Copyright Impinj, Inc. 2013-2015. All rights reserved.
*
*****/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "ipj_util.h"
#include "iri.h"

/* PURPOSE: This example illustrates the use of the tag operations
   Write and Write EPC. */

/* Parameters */
#define IPJ_EXAMPLE_DURATION_MS 250

/* Allocate memory for iri device */
static ipj_iri_device iri_device = { 0 };

static ipj_error write_epc(ipj_iri_device* iri_device, uint16_t* epc)
{
    unsigned int i;
    ipj_error error;
    ipj_key_value key_value[16];
    ipj_key_value key_value_2[16];
    uint32_t key_value_count = 0;
    ipj_key_list key_list;
    ipj_key_list key_list_2;
    uint32_t key_list_count = 0;

    printf("Setting EPC to: ");

    ipj_util_print_epc(epc, 6, false);

    /* Enable tag write */
    key_value_count = 5;

    IPJ_CLEAR_STRUCT(key_list);
    IPJ_CLEAR_STRUCT(key_value);

    key_value[0].key = E_IPJ_KEY_TAG_OPERATION_ENABLE;
    key_value[0].value = true;

    key_value[1].key = E_IPJ_KEY_TAG_OPERATION;
    key_value[1].value = E_IPJ_TAG_OPERATION_TYPE_WRITE;

```

```

key_value[2].key = E_IPJ_KEY_WRITE_MEM_BANK;
key_value[2].value = E_IPJ_MEM_BANK_EPC;

key_value[3].key = E_IPJ_KEY_WRITE_WORD_POINTER;
key_value[3].value = 1;

key_value[4].key = E_IPJ_KEY_WRITE_WORD_COUNT;
key_value[4].value = 7;

key_list_count = 1;
key_list.key = E_IPJ_KEY_WRITE_DATA;
key_list.list_count = 7;

key_list.list[0] = 0x3000;

/* Set last two halfwords of EPC to random numbers*/
for (i = 1; i < 7; i++)
{
    key_list.list[i] = epc[i - 1];
}

error = ipj_bulk_set(
    iri_device,
    &key_value[0],
    key_value_count,
    &key_list,
    key_list_count);
IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_bulk_set");

/* Copy over the keys for the sake of simplicity */
memcpy(&key_value_2, &key_value, sizeof(key_value));
memcpy(&key_list_2, &key_list, sizeof(key_list));

/* Set the desired key list read length*/
key_list_2.length = 7;

error = ipj_bulk_get(
    iri_device,
    key_value_2,
    key_value_count,
    &key_list_2,
    key_list_count);
IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_bulk_get");

printf("\n");
printf("Get Key Value Pairs\n");
for (i = 0; i < key_value_count; i++)
{
    printf(
        "Key: 0x%03x Value: %10d Bank Index:%d Value Index:%d\n",
        key_value_2[i].key,
        key_value_2[i].value,
        key_value_2[i].bank_index,
        key_value_2[i].value_index);
}
printf("\n");

printf("Key List\n");

```

```

printf(
    "Key: 0x%03x\t\t    Bank Index:%d Value Index:%d\t",
    key_list_2.key,
    key_list_2.bank_index,
    key_list_2.value_index);

printf("\n[ ");
for (i = 0; i < key_list_2.list_count; i++)
{
    printf("0x%04X ", key_list_2.list[i]);
}
printf("]\n\n");

return E_IPJ_ERROR_SUCCESS;
}

static ipj_error write_epc_feature(ipj_iri_device* iri_device, uint16_t* epc)
{
    unsigned int i;
    ipj_error error;
    ipj_key_value key_value[16];
    uint32_t key_value_count = 0;
    ipj_key_list key_list;
    uint32_t key_list_count = 0;

    printf("Setting EPC to: ");

    ipj_util_print_epc(epc, 6, false);

    /* Enable tag write */
    key_value_count = 4;

    IPJ_CLEAR_STRUCT(key_value);
    IPJ_CLEAR_STRUCT(key_list);

    key_value[0].key = E_IPJ_KEY_TAG_OPERATION_ENABLE;
    key_value[0].value = true;

    key_value[1].key = E_IPJ_KEY_TAG_OPERATION;
    key_value[1].value = E_IPJ_TAG_OPERATION_TYPE_WRITE_EPC;

    key_value[2].key = E_IPJ_KEY_WRITE_EPC_LENGTH_CONTROL;
    key_value[2].value = E_IPJ_WRITE_EPC_LENGTH_CONTROL_AUTO;

    key_value[3].key = E_IPJ_KEY_WRITE_WORD_COUNT;
    key_value[3].value = 6;

    key_list_count = 1;
    key_list.key = E_IPJ_KEY_WRITE_DATA;
    key_list.list_count = 6;

    /* Set EPC to random numbers*/
    for (i = 0; i < 6; i++)
    {
        key_list.list[i] = epc[i];
    }

    error = ipj_bulk_set(

```

```

        iri_device,
        &key_value[0],
        key_value_count,
        &key_list,
        key_list_count);
    IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_bulk_set");

    return E_IPJ_ERROR_SUCCESS;
}

/* Main */
int main(int argc, char* argv[])
{
    /* Holder for EPC */
    uint16_t epc[6] = { 0 };

    /* Define error code */
    ipj_error error;

    /* Set-Get and Bulk_Set-Get variables */
    uint32_t i = 0;

    IPJ_UTIL_CHECK_USER_INPUT_FOR_COM_PORT_RETURN_ON_ERROR()

    /* Seed random number generator */
    srand((unsigned int) time(NULL));

    /* Common example setup */
    error = ipj_util_setup(&iri_device, argv[1]);
    IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_util_setup");

    /* Bulk_Set-Get example */
    /* Set a tag EPC */

    /* Generate 6 random numbers for EPC code */
    for (i = 0; i < 6; i++)
    {
        epc[i] = rand() % 0xffff;
    }

    /* Write EPC to tag */
    write_epc(&iri_device, epc);

    ipj_util_print_divider('* ', 80);
    printf("EPC Write\n");
    ipj_util_print_divider('- ', 80);

    /* Perform inventory to see results of EPC write */
    ipj_util_perform_inventory(&iri_device, IPJ_EXAMPLE_DURATION_MS);

    /* Generate 6 random numbers for EPC code */
    for (i = 0; i < 6; i++)
    {
        epc[i] = rand() % 0xffff;
    }

    /* Write EPC to tag using WRITE_EPC feature */
    write_epc_feature(&iri_device, epc);
}

```

```

ipj_util_print_divider('* ', 80);
printf("EPC Write Feature\n");
ipj_util_print_divider('- ', 80);

/* Perform inventory to see results of EPC write */
ipj_util_perform_inventory(&iri_device, IPJ_EXAMPLE_DURATION_MS);

/* Common example cleanup */
error = ipj_util_cleanup(&iri_device);
IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_util_cleanup");

return 0;
}

```

IRI_Change_Baudrate Example

The IRI_Change_Baudrate example demonstrates how to change the baud rate of UART communication with an Indy SiP.

IRI_Change_Baudrate - Source Code

IRI_Change_Baudrate source code is provided in IRI_Change_Baudrate.c :

```

/*
*****
*
*          IMPINJ CONFIDENTIAL AND PROPRIETARY
*
* This source code is the sole property of Impinj, Inc. Reproduction or
* utilization of this source code in whole or in part is forbidden without
* the prior written consent of Impinj, Inc.
*
* (c) Copyright Impinj, Inc. 2013-2015. All rights reserved.
*
*****/
#include <stdio.h>
#include <string.h>
#include "ipj_util.h"
#include "iri.h"

/* PURPOSE: This example illustrates the use of the change baud rate feature. */

/* Parameters */
#define IPJ_EXAMPLE_DURATION_MS 250

/* Allocate memory for iri device */
static ipj_iri_device iri_device = { 0 };

/* Main */
int main(int argc, char* argv[])
{
    /* Define error code */
    ipj_error error;

    /* IRI Connection parameters */

```

```

ipj_connection_params connection_params;

IPJ_UTIL_CHECK_USER_INPUT_FOR_COM_PORT_RETURN_ON_ERROR()

/* Common example setup */
error = ipj_util_setup(&iri_device, argv[1]);
IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_util_setup");

printf("Setting Baud to 57600 and performing inventory\n");

connection_params.serial.baudrate = E_IPJ_BAUD_RATE_BR57600;
/* Change Baud rate to 57600 bps */
error = ipj_modify_connection(
    &iri_device,
    E_IPJ_CONNECTION_TYPE_SERIAL,
    &connection_params);
IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_modify_connection");

/* Perform inventory to test results of baud rate change */
error = ipj_util_perform_inventory(&iri_device, IPJ_EXAMPLE_DURATION_MS);
IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_util_perform_inventory");

/* Common example cleanup */
error = ipj_util_cleanup(&iri_device);
IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_util_cleanup");

return 0;
}

```

IRI_Empty Example

The IRI_Empty example is an empty program including the IRI library that can be used as the basis for a program communicating with an Indy SiP.

IRI_Empty - Source Code

IRI_Empty source code is provided in IRI_Empty.c :

```

/*
*****
*
*           IMPINJ CONFIDENTIAL AND PROPRIETARY
*
* This source code is the sole property of Impinj, Inc. Reproduction or
* utilization of this source code in whole or in part is forbidden without
* the prior written consent of Impinj, Inc.
*
* (c) Copyright Impinj, Inc. 2013-2015. All rights reserved.
*
*****/
#include <stdio.h>
#include <string.h>
#include "ipj_util.h"
#include "iri.h"

/* PURPOSE: This file is place holder template to begin a new IRI application. */

```

```

/* Parameters */
#define IPJ_EXAMPLE_DURATION_MS 1000

/* Allocate memory for iri device */
static ipj_iri_device iri_device = { 0 };

/* Main */
int main(int argc, char* argv[])
{
    /* Define error code */
    ipj_error error = E_IPJ_ERROR_SUCCESS;

    IPJ_UTIL_CHECK_USER_INPUT_FOR_COM_PORT_RETURN_ON_ERROR()

    /* Common example setup */
    error = ipj_util_setup(&iri_device, argv[1]);
    IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_util_setup");

    /* YOUR CODE HERE */

    /* Common example cleanup */
    error = ipj_util_cleanup(&iri_device);
    IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_util_cleanup");

    return error;
}

```

IRI_GPIO Example

The IRI_GPIO example demonstrates how to use Indy SiPs' GPIOs as inputs and outputs, including triggering inventories.

IRI_GPIO - Source Code

IRI_GPIO source code is provided in IRI_GPIO.c :

```

/*
*****
*
*           IMPINJ CONFIDENTIAL AND PROPRIETARY
*
* This source code is the sole property of Impinj, Inc. Reproduction or
* utilization of this source code in whole or in part is forbidden without
* the prior written consent of Impinj, Inc.
*
* (c) Copyright Impinj, Inc. 2013-2015. All rights reserved.
*
*****/

/* NOTE: For this example, you will need to jumper GPIO0 to GPIO1 */
#include <stdio.h>
#include <string.h>
#include "ipj_util.h"

```

```

#include "iri.h"
#include "platform.h"

/* PURPOSE: This example illustrates the use GPIOs to initiate an inventory
operation. It also has an example of how to override the stop_handler
versus using the default handler. */

/* Parameters */
#define IPJ_EXAMPLE_DURATION_MS 5000

/* Declare local report handlers */
static ipj_error report_handler(
    ipj_iri_device* iri_device,
    ipj_report_id report_id,
    void* report);

static ipj_error stop_report_handler(
    ipj_iri_device* iri_device,
    ipj_stop_report* ipj_stop_report);

static uint32_t ipj_stopped_flag;

static struct ipj_handler event_handlers[] =
{
    { E_IPJ_HANDLER_TYPE_REPORT, &report_handler }
};

/* Allocate memory for iri device */
static ipj_iri_device iri_device = { 0 };

ipj_error register_handlers(ipj_iri_device* iri_device)
{
    ipj_error error;
    unsigned int i;
    for (i = 0; i < (sizeof(event_handlers) / sizeof(event_handlers[0])); i++)
    {
        error = ipj_register_handler(
            iri_device,
            event_handlers[i].type,
            event_handlers[i].handler);
        if (error)
        {
            return error;
        }
    }
    return E_IPJ_ERROR_SUCCESS;
}

static ipj_error perform_gpio_triggered_inventory(
    ipj_iri_device* iri_device,
    uint32_t timeout_ms)
{
    uint32_t end_time_ms;
    ipj_error error;
    ipj_key_value key_value[16];
    uint32_t key_value_count = 0;

    IPJ_CLEAR_STRUCT(key_value);

```

```

/* GPIO 1 as output resting low*/
key_value[0].key = E_IPJ_KEY_GPIO_MODE;
key_value[0].value = E_IPJ_GPIO_MODE_OUTPUT;
key_value[0].bank_index = 1;
key_value_count++;

key_value[1].key = E_IPJ_KEY_GPIO_STATE;
key_value[1].value = E_IPJ_GPIO_STATE_LO;
key_value[1].bank_index = 1;
key_value_count++;

/* GPIO2 as floating input with attached actions */
key_value[2].key = E_IPJ_KEY_GPIO_MODE;
key_value[2].value = E_IPJ_GPIO_MODE_INPUT_ACTION;
key_value[2].bank_index = 2;
key_value_count++;

key_value[3].key = E_IPJ_KEY_GPIO_STATE;
key_value[3].value = E_IPJ_GPIO_STATE_FLOAT;
key_value[3].bank_index = 2;
key_value_count++;

key_value[4].key = E_IPJ_KEY_GPIO_HI_ACTION;
key_value[4].value = E_IPJ_GPI_ACTION_START_INVENTORY;
key_value[4].bank_index = 2;
key_value_count++;

key_value[5].key = E_IPJ_KEY_GPIO_LO_ACTION;
key_value[5].value = E_IPJ_GPI_ACTION_STOP_INVENTORY;
key_value[5].bank_index = 2;
key_value_count++;

/* Set the keys */
error = ipj_bulk_set(iri_device, &key_value[0], key_value_count, NULL, 0);
IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_bulk_set");

/* start the GPIO set up */
error = ipj_start(iri_device, E_IPJ_ACTION_GPIO);
IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_start E_IPJ_ACTION_GPIO");

/* set GPIO1 high */
error = ipj_set(iri_device, E_IPJ_KEY_GPIO_STATE, 1, 0, E_IPJ_GPIO_STATE_HI);
IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_set E_IPJ_KEY_GPIO_STATE");

/* Set example end time */
end_time_ms = platform_timestamp_ms_handler() + timeout_ms;

/* Run inventory until end time reached */
while (platform_timestamp_ms_handler() < end_time_ms)
{
    /* Call ipj_receive to process tag reports */
    error = ipj_receive(iri_device);
    IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_receive");
}

/* Set GPIO1 Low & Stop inventory */
error = ipj_set(iri_device, E_IPJ_KEY_GPIO_STATE, 1, 0, E_IPJ_GPIO_STATE_LO);
IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_set E_IPJ_KEY_GPIO_STATE");

```

```

error = ipj_stop(iri_device, E_IPJ_ACTION_GPIO);
IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_stop E_IPJ_ACTION_GPIO");

/* Set stop end time */
end_time_ms = platform_timestamp_ms_handler() + timeout_ms;

/* Collect the last few tags and look for the stop report */
while (!ipj_stopped_flag && platform_timestamp_ms_handler() < end_time_ms)
{
    /* Call ipj_receive to process tag reports */
    error = ipj_receive(iri_device);
    IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_receive");
}
return E_IPJ_ERROR_SUCCESS;
}

/* Main */
int main(int argc, char* argv[])
{
    /* Define error code */
    ipj_error error;

    uint32_t end_message_ms;

    IPJ_UTIL_CHECK_USER_INPUT_FOR_COM_PORT_RETURN_ON_ERROR()

    /* Common example setup */
    error = ipj_util_setup(&iri_device, argv[1]);
    IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_util_setup");

    /*
     * Override the report handler so the local stop_handler can be called and
     * stop_perform_gpio_triggered_inventory
     */
    register_handlers(&iri_device);

    /* Display a message to the user for ~5 seconds */
    end_message_ms = platform_timestamp_ms_handler() + 5000;
    printf("Please jump GPIO1 to GPIO2 for this demonstration\n");
    while (platform_timestamp_ms_handler() < end_message_ms)
    {
        /* Do nothing */
    }

    /* Start inventory */
    error = perform_gpio_triggered_inventory(&iri_device, IPJ_EXAMPLE_DURATION_MS);
    IPJ_UTIL_RETURN_ON_ERROR(error, "perform_gpio_triggered_inventory");

    /* Common example cleanup */
    error = ipj_util_cleanup(&iri_device);
    IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_util_cleanup");

    return 0;
}

/* Report handler processes asynchronous reports */
static ipj_error report_handler(
    ipj_iri_device* iri_device,

```

```

    ipj_report_id report_id,
    void* report)
{
    ipj_error error = E_IPJ_ERROR_SUCCESS;
    /* Case statement for each report type */
    switch (report_id)
    {
        case E_IPJ_REPORT_ID_TAG_OPERATION_REPORT:
            error = ipj_util_tag_operation_report_handler(
                iri_device,
                (ipj_tag_operation_report*) report);
            break;
        case E_IPJ_REPORT_ID_STOP_REPORT:
            error = stop_report_handler(iri_device, (ipj_stop_report*) report);
            break;
        case E_IPJ_REPORT_ID_GPIO_REPORT:
            error = ipj_util_gpio_report_handler(
                iri_device,
                (ipj_gpio_report*) report);
            break;
        default:
            printf(
                "%s: REPORT ID: %d NOT HANDLED\n",
                (char*) iri_device->reader_identifier,
                report_id);
            error = E_IPJ_ERROR_GENERAL_ERROR;
            break;
    }
    return error;
}

/* Stop report handler processes asynchronous reports */
static ipj_error stop_report_handler(
    ipj_iri_device* iri_device,
    ipj_stop_report* ipj_stop_report)
{
    if (ipj_stop_report->error == E_IPJ_ERROR_SUCCESS)
    {
        /* Print reader identifier */
        printf("%s: STOPPED\n", (char*) iri_device->reader_identifier);

        /* Set the stopped flag, the stop report does not have any fields that
         * need to be checked */
        ipj_stopped_flag = 1;
    }
    else
    {
        IPJ_UTIL_PRINT_ERROR(ipj_stop_report->error, "stop_report");
    }
    return ipj_stop_report->error;
}

```

IRI_Loader Example

The IRI_Loader example demonstrates how to load a firmware image into an Indy SiP using the IRI host library.

IRI Loader - Source Code

IRI Loader source code is provided in IRI Loader.c :

```

/*
*****
*
*          IMPINJ CONFIDENTIAL AND PROPRIETARY
*
*
* This source code is the sole property of Impinj, Inc.  Reproduction or
* utilization of this source code in whole or in part is forbidden without
* the prior written consent of Impinj, Inc.
*
*
* (c) Copyright Impinj, Inc. 2013-2015. All rights reserved.
*
*****/

#include <stdio.h>

#include <string.h>

#if defined(__GNUC__)
    #include <unistd.h>
#else
    #include <io.h>
#endif

#include "ipj_util.h"

#include "iri.h"

/* PURPOSE: This example illustrates the use of the image loader to download
firmware to the device. */

/* Parameters */

#define IPJ_EXAMPLE_DURATION_MS 1000

```

```
/* Allocate memory for iri device */
static ipj_iri_device iri_device = { 0 };

/* Main */
int main(int argc, char* argv[])
{
    /* Define error code */
    ipj_error error;

    /* Storage buffer for device read */
    uint8_t file_buf[300];
    FILE* image_file_handle;

    int chunk_size;

    /* connection parameters */
    ipj_connection_params params;

    /* High speed flag */
    bool high_speed = false;

    if ((argc < 3) || (argc > 4))
    {
        printf("\n\nUsage:"
            "\n\tIRI_Loader.exe COMx <image_location> [-s] \n"
            "where: \n"
            "\tx is a COM port number\n"
            "\t<image_location> is an absolute path\n"
            "\t-s for high speed 921600 baud update (optional)\n");
    }
}
```

```
    return -1;
}

/* Determine if we're in high speed mode or not */
if ((argc == 4) && (strcmp(argv[3], "-s") == 0))
{
    high_speed = true;
}

/* Common example setup */
error = ipj_util_setup(&iri_device, argv[1]);
IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_util_setup");

/* Put the device in bootloader mode */
error = ipj_reset(&iri_device, E_IPJ_RESET_TYPE_TO_BOOTLOADER);
IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_reset E_IPJ_RESET_TYPE_TO_BOOTLOADER");

if (high_speed)
{
    /* Speed up the upgrade process */

    /* Perform a modify connection command to bring the device
     * and host up to 921600 baud */
    params.serial.baudrate = E_IPJ_BAUD_RATE_BR921600;
    params.serial.parity = E_IPJ_PARITY_PNONE;

    error = ipj_modify_connection(&iri_device,
        E_IPJ_CONNECTION_TYPE_SERIAL, &params);
    if (error)
```

```
    {
        printf("Unable to change baud rate");
        return -1;
    }
}

#if defined(__GNUC__)
    image_file_handle = fopen(argv[2], "rb");
#else
    fopen_s(&image_file_handle, argv[2], "rb");
#endif

if (image_file_handle == NULL)
{
    printf("Unable to open image file\n");
    return -1;
}

/* Get the image chunk size. This is stored in the first 32 bits
 * of the upgrade image */
if (fread(file_buf, 4, 1, image_file_handle) == 0)
{
    printf("Unable to determine chunk size\n");
    return -1;
}

chunk_size = (file_buf[0] & 0xff) | (file_buf[1] << 8) | (file_buf[2] << 16)
             | (file_buf[3] << 24);
```

```
printf("Image chunk size: %d\n", chunk_size);

printf(
    "(%d bytes header | %d bytes payload | 2 bytes CRC)\n",
    12,
    (chunk_size - 2) - 12);

if (chunk_size < 22 || chunk_size > 270)
{
    printf("Invalid chunk size\n");
    return -1;
}

/* For each chunk in the image file, write it to the RS500 */
while (fread(file_buf, chunk_size, 1, image_file_handle) > 0)
{
    error = ipj_flash_handle_loader_block(&iri_device, chunk_size, file_buf);
    IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_flash_handle_loader_block");
}

printf("Image load complete\n");

if (high_speed)
{
    /* Return to our previous baud rate */
    params.serial.baudrate = E_IPJ_BAUD_RATE_BR115200;
    error = ipj_modify_connection(&iri_device,
        E_IPJ_CONNECTION_TYPE_SERIAL, &params);
    if (error)
    {
```

```

        printf("Unable to change baud rate");

        return -1;

    }

}

/* Common example cleanup */

error = ipj_util_cleanup(&iri_device);

IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_util_cleanup");

return 0;

}

```

IRI_Multiple_Readers Example

The IRI_Multiple_Readers example connects to multiple Indy SiPs and performs basic inventory using both.

IRI_Multiple_Readers - Source Code

IRI_Multiple_Readers source code is provided in IRI_Multiple_Readers.c :

```

/*
*****
*
*          IMPINJ CONFIDENTIAL AND PROPRIETARY
*
* This source code is the sole property of Impinj, Inc. Reproduction or
* utilization of this source code in whole or in part is forbidden without
* the prior written consent of Impinj, Inc.
*
* (c) Copyright Impinj, Inc. 2013-2015. All rights reserved.
*
*****/
#include <stdio.h>
#include <string.h>
#include "ipj_util.h"
#include "iri.h"
#include "platform.h"

/* PURPOSE: This example illustrates the use of multiple readers from one
application. */

/* Parameters */
#define IPJ_EXAMPLE_DURATION_MS 1000
#define MAX_READERS 5

/* Declare local report handlers */

```

```

static ipj_error report_handler(
    ipj_iri_device* iri_device,
    ipj_report_id report_id,
    void* report);

static ipj_error stop_report_handler(
    ipj_iri_device* iri_device,
    ipj_stop_report* ipj_stop_report);

/* Allocate memory for iri device */
static ipj_iri_device iri_devices[MAX_READERS];

static uint32_t ipj_stopped_flags[MAX_READERS];

static struct ipj_handler event_handlers[] =
{
    { E_IPJ_HANDLER_TYPE_REPORT, &report_handler }
};

static ipj_error register_handlers(ipj_iri_device* iri_device)
{
    ipj_error error;
    unsigned int i;
    for (i = 0; i < (sizeof(event_handlers) / sizeof(event_handlers[0])); i++)
    {
        error = ipj_register_handler(
            iri_device,
            event_handlers[i].type,
            event_handlers[i].handler);
        if (error)
        {
            return error;
        }
    }
    return E_IPJ_ERROR_SUCCESS;
}

static ipj_error perform_inventory(int num_readers, uint32_t timeout_ms)
{
    uint32_t end_time_ms;
    ipj_error error;
    int i;
    bool all_stopped_flag = false;

    for (i = 0; i < num_readers; i++)
    {
        ipj_stopped_flags[i] = 0x00;
        error = ipj_start(&iri_devices[i], E_IPJ_ACTION_INVENTORY);
        IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_start E_IPJ_ACTION_INVENTORY");
    }

    /* Set example end time */
    end_time_ms = platform_timestamp_ms_handler() + timeout_ms;

    /* Run inventory until end time reached */
    while (platform_timestamp_ms_handler() < end_time_ms)
    {
        for (i = 0; i < num_readers; i++)

```

```

    {
        /* Call ipj_receive to process tag reports */
        error = ipj_receive(&iri_devices[i]);
        IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_receive");
    }
}

for (i = 0; i < num_readers; i++)
{
    /* Stop inventory */
    error = ipj_stop(&iri_devices[i], E_IPJ_ACTION_INVENTORY);
    IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_stop");
}

/* Set stop end time */
end_time_ms = platform_timestamp_ms_handler() + timeout_ms;

while (!all_stopped_flag && platform_timestamp_ms_handler() < end_time_ms)
{
    for (i = 0; i < num_readers; i++)
    {
        /* Collect the last few tags and look for the stop report */
        if (!ipj_stopped_flags[i])
        {
            /* Call ipj_receive to process tag reports */
            error = ipj_receive(&iri_devices[i]);
            IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_receive");
        }
        all_stopped_flag &= ipj_stopped_flags[i];
    }
}

return E_IPJ_ERROR_SUCCESS;
}

/* Main */
int main(int argc, char* argv[])
{
    /* Loop placeholder */
    int i;

    /* Define error code */
    ipj_error error = E_IPJ_ERROR_SUCCESS;

    if (argc < 2)
    {
        printf(
            "\n\nUsage:\t%s COMx COMy...COMz \n\nwhere x/y/z is a COM port number (Maximum 5 read\n\n",
            argv[0]);
        return -1;
    }

    /* Loop through all of the devices we've supplied and set them up */
    for (i = 0; i < argc - 1; i++)
    {
        /* Common example setup */
        error = ipj_util_setup(&iri_devices[i], argv[1 + i]);
        IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_util_setup");
    }
}

```

```

    /*
     * Override the report handler so the local stop_handler can be called
     */
    register_handlers(&iri_devices[i]);
}

/* Start and run inventory with all devices for the specified
 * duration */
error = perform_inventory(argc - 1, IPJ_EXAMPLE_DURATION_MS);
IPJ_UTIL_RETURN_ON_ERROR(error, "perform_inventory");

/* Loop through and disconnect/deinit each iri_device */
for (i = 0; i < argc - 1; i++)
{
    /* Common example cleanup */
    error = ipj_util_cleanup(&iri_devices[i]);
    IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_util_cleanup");
}
return error;
}

/* Report handler processes asynchronous reports */
static ipj_error report_handler(
    ipj_iri_device* iri_device,
    ipj_report_id report_id,
    void* report)
{
    ipj_error error = E_IPJ_ERROR_SUCCESS;
    /* Case statement for each report type */
    switch (report_id)
    {
        case E_IPJ_REPORT_ID_TAG_OPERATION_REPORT:
            error = ipj_util_tag_operation_report_handler(
                iri_device,
                (ipj_tag_operation_report*) report);
            break;
        case E_IPJ_REPORT_ID_STOP_REPORT:
            error = stop_report_handler(iri_device, (ipj_stop_report*) report);
            break;
        case E_IPJ_REPORT_ID_GPIO_REPORT:
            error = ipj_util_gpio_report_handler(
                iri_device,
                (ipj_gpio_report*) report);
            break;
        default:
            printf(
                "%s: REPORT ID: %d NOT HANDLED\n",
                (char*) iri_device->reader_identifier,
                report_id);
            error = E_IPJ_ERROR_GENERAL_ERROR;
            break;
    }
    return error;
}

/* Tag report handler processes asynchronous reports */
static ipj_error stop_report_handler(
    ipj_iri_device* iri_device,

```

```

    ipj_stop_report* ipj_stop_report)
{
    int i;
    if (ipj_stop_report->error == E_IPJ_ERROR_SUCCESS)
    {
        /* Print reader identifier */
        printf("%s: STOPPED\n", (char*) iri_device->reader_identifier);

        /* Set the stopped flag, the stop report does not have any fields that
         * need to be checked */
        for (i = 0; i < MAX_READERS; i++)
        {
            if (&iri_devices[i] == iri_device)
            {
                ipj_stopped_flags[i] = 1;
            }
        }
    }
    else
    {
        printf("IPJ_STOP Error. Error Code:%x\n\n", ipj_stop_report->error);
    }
    return ipj_stop_report->error;
}

```

IRI_Power_Management Example

The IRI_Power_Management example demonstrates changing Indy SiP power modes using the IRI host library.

IRI_Power_Management - Source Code

IRI_Power_Management source code is provided in IRI_Power_Management.c :

```

/*
*****
*
*           IMPINJ CONFIDENTIAL AND PROPRIETARY
*
*
* This source code is the sole property of Impinj, Inc. Reproduction or
* utilization of this source code in whole or in part is forbidden without
* the prior written consent of Impinj, Inc.
*
*
* (c) Copyright Impinj, Inc. 2013-2015. All rights reserved.
*
*****/
#include <stdio.h>
#include <string.h>
#include "ipj_util.h"
#include "iri.h"
#include "platform.h"

/* PURPOSE: This example illustrates the use of the power management modes. */

/* Parameters */
#define IPJ_EXAMPLE_DURATION_MS 1000

```

```

/* Allocate memory for iri device */
static ipj_iri_device iri_device = { 0 };

static ipj_error sleep_ms(uint32_t sleep_time_ms)
{
    uint32_t end_time_ms = platform_timestamp_ms_handler() + sleep_time_ms;
    printf("Sleeping for %d milliseconds\n", sleep_time_ms);

    while (platform_timestamp_ms_handler() < end_time_ms)
        ;
    return E_IPJ_ERROR_SUCCESS;
}

/* Main */
int main(int argc, char* argv[])
{
    /* Define error code */
    ipj_error error = E_IPJ_ERROR_SUCCESS;
    uint32_t val = 0;
    int retries = 10;

    IPJ_UTIL_CHECK_USER_INPUT_FOR_COM_PORT_RETURN_ON_ERROR()

    /* Common example setup */
    error = ipj_util_setup(&iri_device, argv[1]);
    IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_util_setup");

    /* Configure transmit power */
    error = ipj_set_value(&iri_device, E_IPJ_KEY_ANTENNA_TX_POWER, 2300);
    IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_set_value E_IPJ_KEY_ANTENNA_TX_POWER");

    /* Start inventory */
    error = ipj_util_perform_inventory(&iri_device, IPJ_EXAMPLE_DURATION_MS);
    IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_util_perform_inventory");

    /* Put the device into standby mode */
    error = ipj_start(&iri_device, E_IPJ_ACTION_STANDBY);
    IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_start E_IPJ_ACTION_STANDBY");

    /* Sleep for 5 seconds */
    sleep_ms(5000);

    printf("Attempting to wake RS500\n");

    /* Set the receive timeout MS to a lower number (100ms) */
    error = ipj_set_receive_timeout_ms(&iri_device, 100);
    IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_set_receive_timeout_ms");

    /* Spin on IPJ_GET_VALUE (any key will do) until the device responds */
    do
    {
        error = ipj_get_value(&iri_device, E_IPJ_KEY_ANTENNA_TX_POWER, &val);
    } while (error && retries--);
    IPJ_UTIL_RETURN_ON_ERROR(error, "Wake RS500");

    printf("Successfully woke RS500\n");
    printf("Performing Inventory to check for full functionality\n");
}

```

```

/* Reset the timeout to it's default value */
error = ipj_set_receive_timeout_ms(&iri_device, IPJ_DEFAULT_RECEIVE_TIMEOUT_MS);
IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_set_receive_timeout_ms");

/* Perform second inventory */
error = ipj_util_perform_inventory(&iri_device, IPJ_EXAMPLE_DURATION_MS);
IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_util_perform_inventory");

printf("Inventory Successful, putting device into sleep mode\n");
error = ipj_start(&iri_device, E_IPJ_ACTION_SLEEP);
IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_start E_IPJ_ACTION_SLEEP");

printf("Successfully put device into sleep. To wake:\n");
printf("Jump WKUP to VCC -or- press the reset button\n");

/* Disconnect IRI device & close serial port */
error = ipj_disconnect(&iri_device);
IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_disconnect");

/* Deinitialize IRI device */
error = ipj_deinitialize_iri_device(&iri_device);
IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_deinitialize_iri_device");

return error;
}

```

IRI_Select Example

The IRI_Select example demonstrates how to use the Select tag operation to inventory only tags with a certain TID.

This example can be easily modified to select based on EPC as well by changing the *E_IPJ_KEY_SELECT_MEM_BANK* and *E_IPJ_KEY_SELECT_POINTER* keys.

IRI_Select - Source Code

IRI_Select source code is provided in IRI_Select.c :

```

/*
*****
*
*          IMPINJ CONFIDENTIAL AND PROPRIETARY
*
* This source code is the sole property of Impinj, Inc. Reproduction or
* utilization of this source code in whole or in part is forbidden without
* the prior written consent of Impinj, Inc.
*
* (c) Copyright Impinj, Inc. 2013-2015. All rights reserved.
*
*****/
#include <stdio.h>
#include <string.h>
#include "ipj_util.h"
#include "iri.h"

/* PURPOSE: This example illustrates the use of Gen2 Select operation. */

```

```

/* Parameters */
#define IPJ_EXAMPLE_DURATION_MS 250

/* Allocate memory for iri device */
static ipj_iri_device iri_device = { 0 };

/* This select command will cause only tags with a TID starting with
 * 0xe280 to set their select flags */
static ipj_error setup_select_1(ipj_iri_device* iri_device)
{
    ipj_error error = E_IPJ_ERROR_SUCCESS;

    error |= ipj_set(iri_device,
        E_IPJ_KEY_SELECT_ENABLE, 0, 0, true);

    error |= ipj_set(iri_device,
        E_IPJ_KEY_SELECT_TARGET, 0, 0,
        E_IPJ_SELECT_TARGET_SL_FLAG);

    error |= ipj_set(iri_device,
        E_IPJ_KEY_SELECT_ACTION, 0, 0,
        E_IPJ_SELECT_ACTION_ASLINVA_DSLINVB);

    error |= ipj_set(iri_device,
        E_IPJ_KEY_SELECT_MEM_BANK, 0, 0,
        E_IPJ_MEM_BANK_TID);

    error |= ipj_set(iri_device,
        E_IPJ_KEY_SELECT_POINTER, 0, 0, 0x0);

    error |= ipj_set(iri_device,
        E_IPJ_KEY_SELECT_MASK_LENGTH, 0, 0, 16);

    error |= ipj_set(iri_device,
        E_IPJ_KEY_SELECT_MASK_VALUE, 0, 0, 0xE280);

    return error;
}

/* This select command will cause only tags with 1 at bit 19
to maintain their select flag, else their SL flag will de-assert.
This causes an AND effect with the first select.
The 2 selects combined will select Monza4/5 tags*/
static ipj_error setup_select_2(ipj_iri_device* iri_device)
{
    ipj_error error = E_IPJ_ERROR_SUCCESS;

    error |= ipj_set(iri_device,
        E_IPJ_KEY_SELECT_ENABLE, 1, 0, true);

    error |= ipj_set(iri_device,
        E_IPJ_KEY_SELECT_TARGET, 1, 0,
        E_IPJ_SELECT_TARGET_SL_FLAG);

    error |= ipj_set(iri_device,
        E_IPJ_KEY_SELECT_ACTION, 1, 0,
        E_IPJ_SELECT_ACTION_NOTHING_DSLINVB);
}

```

```

error |= ipj_set(iri_device,
E_IPJ_KEY_SELECT_MEM_BANK, 1, 0,
E_IPJ_MEM_BANK_TID);

error |= ipj_set(iri_device,
E_IPJ_KEY_SELECT_POINTER, 1, 0, 19);

error |= ipj_set(iri_device,
E_IPJ_KEY_SELECT_MASK_LENGTH, 1, 0, 1);

/* Note: When the mod of the length is less than 16, the MSBs
of the mask are used. So for 1 bit length, the msb of the
mask (bit 15) is the mask and the LSBs are ignored. */
error |= ipj_set(iri_device,
E_IPJ_KEY_SELECT_MASK_VALUE, 1, 0, 0x8000);

return error;
}

/* Main */
int main(int argc, char* argv[])
{
    /* Define error code */
    ipj_error error;

    IPJ_UTIL_CHECK_USER_INPUT_FOR_COM_PORT_RETURN_ON_ERROR()

    /* Common example setup */
    error = ipj_util_setup(&iri_device, argv[1]);
    IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_util_setup");

    printf("Sending Select Command for Monza4/5 tags (TID = 0xE280.1xxx)\n");
    /* Send Select Command */
    error = setup_select_1(&iri_device);
    IPJ_UTIL_RETURN_ON_ERROR(error, "setup_select_1");

    error = setup_select_2(&iri_device);
    IPJ_UTIL_RETURN_ON_ERROR(error, "setup_select_2");

    /*
    * NOTE: In this example, we target the SL flag, hence setting it
    * here. Your Select use case may or may not require this.
    * Please refer to the GEN2 spec for details
    */
    error = ipj_set_value(&iri_device, E_IPJ_KEY_INVENTORY_SELECT_FLAG, E_IPJ_INVENTORY_SELECT_FLAG_1);
    IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_set_value E_IPJ_KEY_INVENTORY_SELECT_FLAG");

    error = ipj_set_value(&iri_device, E_IPJ_KEY_TAG_OPERATION, 0);
    IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_set_value E_IPJ_KEY_TAG_OPERATION");

    /* Perform inventory to see results of EPC write */
    ipj_util_perform_inventory(&iri_device, IPJ_EXAMPLE_DURATION_MS);

    /* Common example cleanup */
    error = ipj_util_cleanup(&iri_device);
    IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_util_cleanup");

    return 0;
}

```

}

IRI_Test_CW_PRBS Example

The IRI_Test_CW_PRBS example demonstrates how to use some of the IRI host library test commands, such as CW (Continuous Wave) and PRBS (Pseudo-Random Bit Sequence).

IRI_Test_CW_PRBS - Source Code

IRI_Test_CW_PRBS source code is provided in IRI_Test_CW_PRBS.c :

```

/*
*****
*
*           IMPINJ CONFIDENTIAL AND PROPRIETARY
*
*
* This source code is the sole property of Impinj, Inc. Reproduction or
* utilization of this source code in whole or in part is forbidden without
* the prior written consent of Impinj, Inc.
*
* (c) Copyright Impinj, Inc. 2013-2015. All rights reserved.
*
*****/
#include <stdio.h>
#include <string.h>
#include "ipj_util.h"
#include "iri.h"
#include "platform.h"

/* PURPOSE: This example illustrates the use of the test commands to control
the CW (Continuous Wave) and PRBS (Pseudo-Random Bit Sequence) functions. */

/* Parameters */
#define IPJ_EXAMPLE_DURATION_MS 10000

/* Allocate memory for iri device */
static ipj_iri_device iri_device = { 0 };

/* Main */
int main(int argc, char* argv[])
{
    /* Define error code */
    ipj_error error;

    IPJ_UTIL_CHECK_USER_INPUT_FOR_COM_PORT_RETURN_ON_ERROR()

    /* Common example setup */
    error = ipj_util_setup(&iri_device, argv[1]);
    IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_util_setup");

    /* Set frequency to 912.25 MHz (specified in kHz)*/
    ipj_util_test_command(&iri_device, E_IPJ_TEST_ID_SET_FREQUENCY, 912250, 0, 0, 0);

    /* Enable CW */
    ipj_util_test_command(&iri_device, E_IPJ_TEST_ID_CW_CONTROL, 1, 0, 0, 0);

```

```

/* Leave CW on for 1 second */
platform_sleep_ms_handler(1000);

/* Disable CW */
ipj_util_test_command(&iri_device, E_IPJ_TEST_ID_CW_CONTROL, 0, 0, 0, 0);

/* Set frequency to 912.75 MHz (specified in kHz)*/
ipj_util_test_command(&iri_device, E_IPJ_TEST_ID_SET_FREQUENCY, 912750, 0, 0, 0);

/* Enable PRBS */
ipj_util_test_command(&iri_device, E_IPJ_TEST_ID_PRBS_CONTROL, 1, 0, 0, 0);

/* Leave PRBS on for 1 second */
platform_sleep_ms_handler(1000);

/* Disable PRBS */
ipj_util_test_command(&iri_device, E_IPJ_TEST_ID_PRBS_CONTROL, 0, 0, 0, 0);

/* Common example cleanup */
error = ipj_util_cleanup(&iri_device);
IPJ_UTIL_RETURN_ON_ERROR(error, "ipj_util_cleanup");

return 0;
}

```

Platform - Source Code

The platform source code is common between all of the examples, although it differs for each host platform.

Platform source code is provided in the `platform_*.c` files.

For example, here is `platform_win32.c`:

```

/*
*****
*
*           IMPINJ CONFIDENTIAL AND PROPRIETARY
*
* This source code is the sole property of Impinj, Inc. Reproduction or
* utilization of this source code in whole or in part is forbidden without
* the prior written consent of Impinj, Inc.
*
* (c) Copyright Impinj, Inc. 2013-2015. All rights reserved.
*
*****/

#include <windows.h>
#include <stdint.h>
#include <stdio.h>
#include "platform.h"
#include "iri.h"

/* PLATFORM_OPEN_PORT_HANDLER */
uint32_t platform_open_port_handler(IPJ_READER_CONTEXT* reader_context,
                                   IPJ_READER_IDENTIFIER reader_identifier,
                                   ipj_connection_type connection_type,
                                   ipj_connection_params* params)
{

```

```

HANDLE h_port;
DCB dcb;
COMMTIMEOUTS commtimeouts;

uint32_t result;
char port_name[30];

/* we currently only support serial */
(void) connection_type;

/* Map reader_identifier to COM port name */
#if defined(__GNUC__)
    strcpy(port_name, "\\\\.\\.\\.\\.\\.\\.\\.\\.");
    strcat(port_name, (char*)reader_identifier);
#else /* Visual Studio/Intel/etc */
    strcpy_s(port_name, sizeof(port_name), "\\\\.\\.\\.\\.\\.\\.\\.\\.");
    strcat_s(port_name, sizeof(port_name), (char*)reader_identifier);
#endif

/* Open serial port */
h_port = CreateFileA(port_name, GENERIC_READ | GENERIC_WRITE, 0, 0, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);

/* Return error if invalid handle */
if ((h_port == INVALID_HANDLE_VALUE))
{
    return 0;
}

/* Get default serial port parameters */
dcb.DCBlength = sizeof(dcb);
result = GetCommState(h_port, &dcb);

/* Return error if unsuccessful getting serial port parameters */
if (!result)
{
    return 0;
}

/* Modify serial port parameters */
if (!params)
{
    return 0;
}
else
{
    dcb.BaudRate = params->serial.baudrate;
    switch(params->serial.parity)
    {
        case E_IPJ_PARITY_PODD:
        {
            dcb.Parity = ODDPARITY;
            break;
        }
        case E_IPJ_PARITY_PEVEN:
        {
            dcb.Parity = EVENPARITY;
            break;
        }
    }
}

```

```

        default: /* Fall through */
        case E_IPJ_PARITY_PNONE:
        {
            dcb.Parity = NOPARITY;
            break;
        }
    }
}
dcb.ByteSize = 8;
dcb.StopBits = ONESTOPBIT;
dcb.fParity = false;
dcb.fOutX = false;
dcb.fInX = false;
dcb.fNull = false;
dcb.fDtrControl = DTR_CONTROL_ENABLE;
dcb.fRtsControl = RTS_CONTROL_ENABLE;

/* Set serial port parameters */
result = SetCommState(h_port, &dcb);

/* Return error if unsuccessful setting serial port parameters */
if (!result)
{
    return 0;
}

/* Get timeout values */
result = GetCommTimeouts(h_port, &commtimeouts);

// Return error if unsuccessful getting serial port timeout parameters */
if (!result)
{
    return 0;
}

/* Modify timeout values */
commtimeouts.ReadIntervalTimeout = 0;
commtimeouts.ReadTotalTimeoutConstant = PLATFORM_DEFAULT_READ_TIMEOUT_MS;
commtimeouts.ReadTotalTimeoutMultiplier = 0;
commtimeouts.WriteTotalTimeoutConstant = PLATFORM_DEFAULT_WRITE_TIMEOUT_MS;
commtimeouts.WriteTotalTimeoutMultiplier = 0;

/* Set timeout values */
result = SetCommTimeouts(h_port, &commtimeouts);

if(!result)
{
    return 0;
}

/* Assign serial port handle to reader context */
*reader_context = (IPJ_READER_CONTEXT)h_port;
return 1;
}

/* PLATFORM_CLOSE_PORT_HANDLER */
uint32_t platform_close_port_handler(IPJ_READER_CONTEXT reader_context)
{

```

```

uint32_t result;

/* Close reader handle */
result = CloseHandle((HANDLE) reader_context);

if(!result)
{
    return 0;
}

return 1;
}

/*
 * PLATFORM_TRANSMIT_HANDLER
 *
 * Return: 1: Success, 0:Fail
 */
uint32_t platform_transmit_handler(IPJ_READER_CONTEXT reader_context, uint8_t* message_buffer, uint16_t* number_bytes_transmitted)
{
    uint32_t result;
    DWORD dw_bytes_written = 0;

    /* Write serial port */
    result = WriteFile((HANDLE) reader_context, message_buffer, buffer_size, &dw_bytes_written, NULL);
    if (!result)
    {
        /* Fail */
        return 0;
    }
    else
    {
        /* Success */
        /* Assign number of bytes transmitted */
        *number_bytes_transmitted = (uint16_t) dw_bytes_written;
        return 1;
    }
}

/* PLATFORM_RECEIVE_HANDLER */
uint32_t platform_receive_handler(IPJ_READER_CONTEXT reader_context, uint8_t* message_buffer, uint16_t* number_bytes_received, uint16_t timeout_ms)
{
    uint32_t result;
    DWORD dw_bytes_received = 0;
    COMMTIMEOUTS commtimeouts;

    /* Get timeout values */
    result = GetCommTimeouts((HANDLE) reader_context, &commtimeouts);

    /* Return error if unsuccessful getting serial port timeout parameters */
    if (!result)
    {
        return 0;
    }

    /* Modify timeout values */

```

```

if (timeout_ms == 0)
{
    commtimeouts.ReadIntervalTimeout = MAXDWORD;
    commtimeouts.ReadTotalTimeoutConstant = 1;
}
else
{
    commtimeouts.ReadIntervalTimeout = timeout_ms;
    commtimeouts.ReadTotalTimeoutConstant = 1;
}

/* Set timeout values */
result = SetCommTimeouts((HANDLE)reader_context, &commtimeouts);

/* Return if error if unsuccessful setting timeout parameters */
if(!result)
{
    return 0;
}

/* Read serial port */
result = ReadFile((HANDLE)reader_context, message_buffer, buffer_size, &dw_bytes_received, NULL);
if (result)
{
    /* Assign bytes received */
    *number_bytes_received = (uint16_t)dw_bytes_received;
    return 1;
}
else
{
    return 0;
}
}

/* PLATFORM_TIMESTAMP_HANDLER */
uint32_t platform_timestamp_ms_handler()
{
    DWORD timestamp_ms = timeGetTime();

    return (uint32_t)timestamp_ms;
}

void platform_sleep_ms_handler(uint32_t milliseconds)
{
    Sleep(milliseconds);
}

uint32_t platform_flush_port_handler(IPJ_READER_CONTEXT reader_context)
{
    int result;

    result = PurgeComm((HANDLE)reader_context, PURGE_RXCLEAR | PURGE_TXCLEAR);

    return result == 0 ? 1 : 0;
}

uint32_t platform_reset_pin_handler(IPJ_READER_CONTEXT reader_context,
                                   bool enable)

```

```

{
    int action = enable ? SETDTR : CLRDTR;
    int result = EscapeCommFunction((HANDLE) reader_context, action);
    return result == 0 ? 1 : 0;
}

uint32_t platform_wakeup_pin_handler(IPJ_READER_CONTEXT reader_context,
                                     bool enable)
{
    int action = enable ? SETRTS : CLRRTS;
    int result = EscapeCommFunction((HANDLE) reader_context, action);
    return result == 0 ? 1 : 0;
}

uint32_t platform_modify_connection_handler(IPJ_READER_CONTEXT reader_context,
                                           ipj_connection_type connection_type,
                                           ipj_connection_params* params)
{
    DCB dcb;
    int result;

    switch(connection_type)
    {
        /* For all serial operations */
        case E_IPJ_CONNECTION_TYPE_SERIAL:
        {
            /* Get serial port parameters */
            result = GetCommState((HANDLE) reader_context, &dcb);
            if (!result)
            {
                return 1;
            }
            /* Set the new baudrate */
            dcb.BaudRate = (DWORD)params->serial.baudrate;

            switch(params->serial.parity)
            {
                case E_IPJ_PARITY_PODD:
                {
                    dcb.Parity = ODDPARITY;
                    break;
                }
                case E_IPJ_PARITY_PEVEN:
                {
                    dcb.Parity = EVENPARITY;
                    break;
                }
                default: /* Fall through */
                case E_IPJ_PARITY_PNONE:
                {
                    dcb.Parity = NOPARITY;
                    break;
                }
            }
        }

        result = SetCommState((HANDLE) reader_context, &dcb);
        if (!result)

```

```

        {
            return 1;
        }
        return 0;
    }
    default:
    {
        return 0;
    }
}
}

```

1.5.7 IRI Configuration Examples

Configure the Indy SiP using *ipj_set*, *ipj_set_value*, or *ipj_bulk_set*. The *ipj_set*, *ipj_set_value*, and *ipj_bulk_set* functions allow the user to write data to keys, or registers, within the Indy SiP. See the *Key Codes* section for a complete list of keys.

The *ipj_set*, *ipj_set_value*, and *ipj_bulk_set* functions write configuration data to the Indy SiP. The Indy SiP applies the configuration settings when the *ipj_start* function is called.

Current settings may be read from the Indy SiP by using *ipj_get*, *ipj_get_value*, or *ipj_bulk_get*.

Warning: Setting keys will persist their value until they are changed or the device is reset. It is important to explicitly configure keys for each operation in a manner which are deterministic. For example, if a Tag operation is enabled the device will perform the configured tag operation on each start command until disabled.

Radio Control

Regulatory Region

Configure the regulatory region using the *E_IPJ_KEY_REGION_ID* key.

Note: The RS500 GX SKUs supports regions in the 902-928 MHz frequency range, including the FCC. The RS500 EU SKUs supports regions in the 865-868 MHz frequency range, including ETSI. Each RS2000 SKU supports 2 different frequency ranges, specified in the device datasheet.

The following code snippet sets the regulatory region to ETSI EN 302 208 v1.4.1:

```
ipj_set_value(&iri_device, E_IPJ_KEY_REGION_ID, E_IPJ_REGION_ETSI_EN_302_208_V1_4_1);
```

Transmit Power

Configure the transmit power using the *E_IPJ_KEY_ANTENNA_TX_POWER* key.

The following code snippet sets the transmit power to 20.00 dBm:

```
ipj_set_value(&iri_device, E_IPJ_KEY_ANTENNA_TX_POWER, 2000);
```

RF Mode

Configure the RF Mode using the `E_IPJ_KEY_RF_MODE` key.

The following code snippet sets the RF Mode to 3:

```
ipj_set_value(&iri_device, E_IPJ_KEY_RF_MODE, 3);
```

Antenna Switching

Configure the antenna switching sequence using the `E_IPJ_KEY_ANTENNA_SEQUENCE` key.

The following code snippet sets the antenna switching sequence to 1,2,4,3,2:

```
ipj_key_list    key_list;
uint32_t       key_list_count;

memset(&key_list, 0, sizeof(key_list));

key_list_count = 1;
key_list.key = E_IPJ_KEY_ANTENNA_SEQUENCE;

key_list.list_count = 16;
key_list.list[0] = 1;
key_list.list[1] = 2;
key_list.list[2] = 4;
key_list.list[3] = 3;
key_list.list[4] = 2;
// All other list items initialized to 0 by memset

ipj_bulk_set(&iri_device, 0, 0, &key_list, key_list_count);
```

RFID Configuration

Indy SiPs manage Tag populations using three basic operations:

- *Select*: *Select* operations may be applied to select a particular Tag population based on user-specified criteria.
- *Inventory*: The Indy SiP identifies individual Tags during Inventory. Tag EPCs are reported in *Tag Operation Reports*.
- *Access*: The Indy SiP communicates with individual Tags using Access operations. Access operations include the following: reading tag memory, writing tag memory, locking tag memory, blockpermalocking tag memory, and killing a tag. Results of Access operations are reported in *Tag Operation Reports*.

Details of the Select, Inventory, and Access operations may be found in the UHF Gen2 specification: [GS1/EPCglobal Radio-Frequency Identity Protocol Generation-2 UHF RFID Protocol for Communications at 860 MHz - 960 MHz. Version 1.20](#)

Users configure the Select, Inventory, and Access parameters prior to calling the `ipj_start` function. In response to the `ipj_start` function, the Indy SiP will transmit the appropriate Select, Inventory, and Access commands. Results of Inventory and Access operations are reported in *Tag Operation Reports*.

Select Parameters

The Select operation is optional.

Configure Select operation using the following keys (see the UHF Gen2 specification for details about the *Select* command):

- *E_IPJ_KEY_SELECT_ENABLE*
- *E_IPJ_KEY_SELECT_TARGET*
- *E_IPJ_KEY_SELECT_ACTION*
- *E_IPJ_KEY_SELECT_MEM_BANK*
- *E_IPJ_KEY_SELECT_POINTER*
- *E_IPJ_KEY_SELECT_MASK_LENGTH*
- *E_IPJ_KEY_SELECT_MASK_VALUE*

Warning: Setting the enable to true will persist until changed. If no longer desired, the user must configure the enable back to false.

Example Select operation configuration

```
ipj_set_value(&iri_device, E_IPJ_KEY_SELECT_ENABLE, true);
ipj_set_value(&iri_device, E_IPJ_KEY_SELECT_TARGET, E_IPJ_SELECT_TARGET_SL_FLAG);
ipj_set_value(&iri_device, E_IPJ_KEY_SELECT_ACTION, E_IPJ_SELECT_ACTION_AS LINVA_DSLINVB);
ipj_set_value(&iri_device, E_IPJ_KEY_SELECT_MEM_BANK, E_IPJ_MEM_BANK_EPC);
ipj_set_value(&iri_device, E_IPJ_KEY_SELECT_POINTER, 0x20);
ipj_set_value(&iri_device, E_IPJ_KEY_SELECT_MASK_LENGTH, 96);
ipj_set(&iri_device, E_IPJ_KEY_SELECT_MASK_VALUE, 0, 0, 0x0123);
ipj_set(&iri_device, E_IPJ_KEY_SELECT_MASK_VALUE, 0, 1, 0x4567);
ipj_set(&iri_device, E_IPJ_KEY_SELECT_MASK_VALUE, 0, 2, 0x89AB);
ipj_set(&iri_device, E_IPJ_KEY_SELECT_MASK_VALUE, 0, 3, 0xCDEF);
ipj_set(&iri_device, E_IPJ_KEY_SELECT_MASK_VALUE, 0, 4, 0xFFFF);
ipj_set(&iri_device, E_IPJ_KEY_SELECT_MASK_VALUE, 0, 5, 0x0001);
```

Enabling the Select operation results in the Indy SiP sending Select command before each Inventory round. It is also necessary to configure Inventory to identify and report selected tags (based on the use case and select target selected in the Select Command):

```
ipj_set_value(&iri_device, E_IPJ_KEY_INVENTORY_SELECT_FLAG, E_IPJ_INVENTORY_SELECT_FLAG_SL);
```

After the Indy SiP is configured:

- Start inventory using *ipj_start*
- Process *Tag Operation Reports* in report handler

Inventory Parameters

Configuration of the Inventory parameters is not required. The Indy SiP defaults to a standard Inventory configuration. Recommend users configure Inventory parameters for specific applications.

The following keys control the Inventory operation (see the UHF Gen2 specification for details):

- *E_IPJ_KEY_INVENTORY_TAG_POPULATION*
- *E_IPJ_KEY_INVENTORY_SELECT_FLAG*
- *E_IPJ_KEY_INVENTORY_SESSION*
- *E_IPJ_KEY_INVENTORY_SEARCH_MODE*

Example Inventory parameter configuration

```
ipj_set_value(&iri_device, E_IPJ_KEY_INVENTORY_TAG_POPULATION, 10);
ipj_set_value(&iri_device, E_IPJ_KEY_INVENTORY_SELECT_FLAG, E_IPJ_INVENTORY_SELECT_FLAG_ALL_SL);
ipj_set_value(&iri_device, E_IPJ_KEY_INVENTORY_SESSION, 2);
ipj_set_value(&iri_device, E_IPJ_KEY_INVENTORY_SEARCH_MODE, E_IPJ_INVENTORY_SEARCH_MODE_DUAL_TARGET);
```

The following keys enable Impinj Inventory extensions:

- *E_IPJ_KEY_FAST_ID_ENABLE*
- *E_IPJ_KEY_TAG_FOCUS_ENABLE*

```
ipj_set_value(&iri_device, E_IPJ_KEY_FAST_ID_ENABLE, true);
ipj_set_value(&iri_device, E_IPJ_KEY_TAG_FOCUS_ENABLE, true);
```

After the Indy SiP is configured:

- Start inventory using *ipj_start*
- Process *Tag Operation Reports* in report handler

Access Parameters

Access operations enable users to read Tag memory, write Tag memory, lock Tag memory and kill a Tag.

Users configure the Indy SiP for the desired read, write, lock, blockpermalock, or kill operation. The Indy SiP will perform the desired Access operation on each Tag that is inventoried. The results are provided in the *Tag Operation Reports*.

Warning: Setting the enable to true will persist until changed. If no longer desired, the user must configure the enable back to false.

Read The following keys enable Read operation (see the UHF Gen2 specification for details about the *Read* command):

- *E_IPJ_KEY_TAG_OPERATION_ENABLE*
- *E_IPJ_KEY_TAG_OPERATION*
- *E_IPJ_KEY_ACCESS_PASSWORD* (optional, for secure access)
- *E_IPJ_KEY_READ_MEM_BANK*
- *E_IPJ_KEY_READ_WORD_POINTER*
- *E_IPJ_KEY_READ_WORD_COUNT*

Example Read configuration:

```
ipj_set_value(&iri_device, E_IPJ_KEY_TAG_OPERATION_ENABLE, true);
ipj_set_value(&iri_device, E_IPJ_KEY_TAG_OPERATION, E_IPJ_TAG_OPERATION_TYPE_READ);
ipj_set_value(&iri_device, E_IPJ_KEY_READ_MEM_BANK, E_IPJ_MEM_BANK_TID);
ipj_set_value(&iri_device, E_IPJ_KEY_READ_WORD_POINTER, 0x00);
ipj_set_value(&iri_device, E_IPJ_KEY_READ_WORD_COUNT, 2);
```

After the Indy SiP is configured:

- Start inventory using *ipj_start*
- Process *Tag Operation Reports* to view read data

Write The following keys enable Write operation (see the UHF Gen2 specification for details about the *Write* command):

- *E_IPJ_KEY_TAG_OPERATION_ENABLE*
- *E_IPJ_KEY_TAG_OPERATION*
- *E_IPJ_KEY_ACCESS_PASSWORD* (optional, for secure access)
- *E_IPJ_KEY_WRITE_MEM_BANK*
- *E_IPJ_KEY_WRITE_WORD_POINTER*
- *E_IPJ_KEY_WRITE_WORD_COUNT*
- *E_IPJ_KEY_WRITE_DATA*

Example Write configuration:

```
ipj_set_value(&iri_device, E_IPJ_KEY_TAG_OPERATION_ENABLE, true);
ipj_set_value(&iri_device, E_IPJ_KEY_TAG_OPERATION, E_IPJ_TAG_OPERATION_TYPE_WRITE);
ipj_set_value(&iri_device, E_IPJ_KEY_WRITE_MEM_BANK, E_IPJ_MEM_BANK_EPC);
ipj_set_value(&iri_device, E_IPJ_KEY_WRITE_WORD_POINTER, 0x01);
ipj_set_value(&iri_device, E_IPJ_KEY_WRITE_WORD_COUNT, 7);
ipj_set(&iri_device, E_IPJ_KEY_WRITE_DATA, 0, 0, 0x3000);
ipj_set(&iri_device, E_IPJ_KEY_WRITE_DATA, 0, 1, 0xAAAA);
ipj_set(&iri_device, E_IPJ_KEY_WRITE_DATA, 0, 2, 0x5555);
ipj_set(&iri_device, E_IPJ_KEY_WRITE_DATA, 0, 3, 0x0123);
ipj_set(&iri_device, E_IPJ_KEY_WRITE_DATA, 0, 4, 0x4567);
ipj_set(&iri_device, E_IPJ_KEY_WRITE_DATA, 0, 5, 0x89AB);
ipj_set(&iri_device, E_IPJ_KEY_WRITE_DATA, 0, 6, 0xCDEF);
```

Alternatively, `ipj_bulk_set` can be used to configure the Indy SiP to write tag memory with a single function call

```
ipj_key_value    key_value[16];
uint32_t        key_value_count;
ipj_key_list     key_list;
uint32_t        key_list_count;

memset(key_value, 0, sizeof(key_value));
memset(&key_list, 0, sizeof(key_list));

key_value_count = 5;

key_value[0].key = E_IPJ_KEY_TAG_OPERATION_ENABLE;
key_value[0].value = true;

key_value[1].key = E_IPJ_KEY_TAG_OPERATION;
key_value[1].value = E_IPJ_TAG_OPERATION_TYPE_WRITE;

key_value[2].key = E_IPJ_KEY_WRITE_MEM_BANK;
key_value[2].value = E_IPJ_MEM_BANK_EPC;

key_value[3].key = E_IPJ_KEY_WRITE_WORD_POINTER;
key_value[3].value = 1;

key_value[4].key = E_IPJ_KEY_WRITE_WORD_COUNT;
key_value[4].value = 7;

key_list_count = 1;
key_list.key = E_IPJ_KEY_WRITE_DATA;
```

```

key_list.list_count = 7;
key_list.list[0] = 0x3000;
key_list.list[1] = 0xAAAA;
key_list.list[2] = 0x5555;
key_list.list[3] = 0x0123;
key_list.list[4] = 0x4567;
key_list.list[5] = 0x89AB;
key_list.list[6] = 0xCDEF;

ipj_bulk_set(&iri_device, &key_value[0], key_value_count, &key_list, key_list_count);

```

After the Indy SiP is configured:

- Start inventory using *ipj_start*
- Process *Tag Operation Reports* to confirm successful writes

Write EPC The Write EPC operation writes Tag EPC memory and optionally updates the PC word. When the Write EPC operation is selected, the Indy SiP automatically selects the EPC memory bank and EPC word pointer. Additionally, the Indy SiP updates the PC word according to the *...EPC_LENGTH_CONTROL* and *...AFI_CONTROL* key values. The following keys enable Write EPC operation (see the UHF Gen2 specification for details):

- *E_IPJ_KEY_TAG_OPERATION_ENABLE*
- *E_IPJ_KEY_TAG_OPERATION*
- *E_IPJ_KEY_ACCESS_PASSWORD* (optional, for secure access)
- *E_IPJ_KEY_WRITE_EPC_LENGTH_CONTROL*
- *E_IPJ_KEY_WRITE_EPC_LENGTH_VALUE*
- *E_IPJ_KEY_WRITE_EPC_AFI_CONTROL*
- *E_IPJ_KEY_WRITE_EPC_AFI_VALUE*
- *E_IPJ_KEY_WRITE_WORD_COUNT*
- *E_IPJ_KEY_WRITE_DATA*

Example Write EPC configuration:

```

ipj_set_value(&iri_device, E_IPJ_KEY_TAG_OPERATION_ENABLE, true);
ipj_set_value(&iri_device, E_IPJ_KEY_TAG_OPERATION, E_IPJ_TAG_OPERATION_TYPE_WRITE_EPC);
ipj_set_value(&iri_device, E_IPJ_KEY_WRITE_EPC_LENGTH_CONTROL, E_IPJ_WRITE_EPC_LENGTH_CONTROL_AUTO);
ipj_set_value(&iri_device, E_IPJ_KEY_WRITE_EPC_AFI_CONTROL, 0);
ipj_set_value(&iri_device, E_IPJ_KEY_WRITE_EPC_AFI_VALUE, 0);
ipj_set_value(&iri_device, E_IPJ_KEY_WRITE_WORD_COUNT, 6);
ipj_set(&iri_device, E_IPJ_KEY_WRITE_DATA, 0, 0, 0xAAAA);
ipj_set(&iri_device, E_IPJ_KEY_WRITE_DATA, 0, 1, 0x5555);
ipj_set(&iri_device, E_IPJ_KEY_WRITE_DATA, 0, 2, 0x0123);
ipj_set(&iri_device, E_IPJ_KEY_WRITE_DATA, 0, 3, 0x4567);
ipj_set(&iri_device, E_IPJ_KEY_WRITE_DATA, 0, 4, 0x89AB);
ipj_set(&iri_device, E_IPJ_KEY_WRITE_DATA, 0, 5, 0xCDEF);

```

After the Indy SiP is configured:

- Start inventory using *ipj_start*
- Process *Tag Operation Reports* to confirm successful writes

Lock The following keys enable the Lock operation (see the UHF Gen2 specification for details about the *Lock* command):

- *E_IPJ_KEY_TAG_OPERATION_ENABLE*
- *E_IPJ_KEY_TAG_OPERATION*
- *E_IPJ_KEY_ACCESS_PASSWORD* (optional, for secure access)
- *E_IPJ_KEY_LOCK_PAYLOAD*

Example Lock configuration:

```
ipj_set_value(&iri_device, E_IPJ_KEY_TAG_OPERATION_ENABLE, true);
ipj_set_value(&iri_device, E_IPJ_KEY_TAG_OPERATION, E_IPJ_TAG_OPERATION_TYPE_LOCK);
ipj_set_value(&iri_device, E_IPJ_KEY_LOCK_PAYLOAD, 0xFFFFF);
```

After the Indy SiP is configured:

- Start inventory using *ipj_start*
- Process *Tag Operation Reports* to confirm successful lock

Blockpermalock The following keys enable the Blockpermalock operation (see the UHF Gen2 specification for details about the *Blockpermalock* command):

- *E_IPJ_KEY_TAG_OPERATION_ENABLE*
- *E_IPJ_KEY_TAG_OPERATION*
- *E_IPJ_KEY_ACCESS_PASSWORD* (optional, for secure access)
- *E_IPJ_KEY_BLOCKPERMALOCK_ACTION*
- *E_IPJ_KEY_BLOCKPERMALOCK_MEM_BANK*
- *E_IPJ_KEY_BLOCKPERMALOCK_BLOCK_POINTER*
- *E_IPJ_KEY_BLOCKPERMALOCK_BLOCK_RANGE*
- *E_IPJ_KEY_BLOCKPERMALOCK_MASK*

Example Blockpermalock configuration:

```
ipj_set_value(&iri_device, E_IPJ_KEY_TAG_OPERATION_ENABLE, true);
ipj_set_value(&iri_device, E_IPJ_KEY_TAG_OPERATION, E_IPJ_TAG_OPERATION_TYPE_BLOCKPERMALOCK);
ipj_set_value(&iri_device, E_IPJ_KEY_BLOCKPERMALOCK_ACTION, E_IPJ_BLOCKPERMALOCK_ACTION_PERMALOCK);
ipj_set_value(&iri_device, E_IPJ_KEY_BLOCKPERMALOCK_MEM_BANK, E_IPJ_MEM_BANK_USER);
ipj_set_value(&iri_device, E_IPJ_KEY_BLOCKPERMALOCK_BLOCK_POINTER, 0x00);
ipj_set_value(&iri_device, E_IPJ_KEY_BLOCKPERMALOCK_BLOCK_RANGE, 1);
ipj_set_value(&iri_device, E_IPJ_KEY_BLOCKPERMALOCK_MASK, 0x1);
```

After the Indy SiP is configured:

- Start inventory using *ipj_start*
- Process *Tag Operation Reports* to confirm successful blockpermalock

Kill The following keys enable the Kill operation (see the UHF Gen2 specification for details about the *Kill* procedure):

- *E_IPJ_KEY_TAG_OPERATION_ENABLE*
- *E_IPJ_KEY_TAG_OPERATION*

- *E_IPJ_KEY_KILL_PASSWORD*

Example Kill configuration:

```
ipj_set_value(&iri_device, E_IPJ_KEY_TAG_OPERATION_ENABLE, true);
ipj_set_value(&iri_device, E_IPJ_KEY_TAG_OPERATION, E_IPJ_TAG_OPERATION_TYPE_KILL);
ipj_set_value(&iri_device, E_IPJ_KEY_KILL_PASSWORD, 0x01234567);
```

After the Indy SiP is configured:

- Start inventory using *ipj_start*
- Process *Tag Operation Reports* to confirm success

QT Configuration The following keys enable the QT operation:

- *E_IPJ_KEY_TAG_OPERATION_ENABLE*
- *E_IPJ_KEY_TAG_OPERATION*
- *E_IPJ_KEY_ACCESS_PASSWORD* (optional, for secure access)
- *E_IPJ_KEY_QT_ACTION*
- *E_IPJ_KEY_QT_PERSISTENCE*
- *E_IPJ_KEY_QT_DATA_PROFILE*
- *E_IPJ_KEY_QT_ACCESS_RANGE*
- *E_IPJ_KEY_QT_TAG_OPERATION*

Example QT configuration:

```
ipj_set_value(&iri_device, E_IPJ_KEY_TAG_OPERATION_ENABLE, true);
ipj_set_value(&iri_device, E_IPJ_KEY_TAG_OPERATION, E_IPJ_TAG_OPERATION_TYPE_QT);
ipj_set_value(&iri_device, E_IPJ_KEY_QT_ACTION, E_IPJ_QT_ACTION_READ);
```

After the Indy SiP is configured:

- Start inventory using *ipj_start*
- Process tag operation reports in report handler to confirm success

Example QT configuration to read public EPC:

```
ipj_set_value(&iri_device, E_IPJ_KEY_TAG_OPERATION_ENABLE, true);
ipj_set_value(&iri_device, E_IPJ_KEY_TAG_OPERATION, E_IPJ_TAG_OPERATION_TYPE_QT);
ipj_set_value(&iri_device, E_IPJ_KEY_QT_ACTION, E_IPJ_QT_ACTION_WRITE);
ipj_set_value(&iri_device, E_IPJ_KEY_QT_PERSISTENCE, E_IPJ_QT_PERSISTENCE_TEMPORARY);
ipj_set_value(&iri_device, E_IPJ_KEY_QT_DATA_PROFILE, E_IPJ_QT_DATA_PROFILE_PUBLIC);
ipj_set_value(&iri_device, E_IPJ_KEY_QT_ACCESS_RANGE, E_IPJ_QT_ACCESS_RANGE_NORMAL);
ipj_set_value(&iri_device, E_IPJ_KEY_QT_TAG_OPERATION, E_IPJ_TAG_OPERATION_TYPE_READ);
ipj_set_value(&iri_device, E_IPJ_KEY_READ_MEM_BANK, E_IPJ_MEM_BANK_EPC);
ipj_set_value(&iri_device, E_IPJ_KEY_READ_WORD_POINTER, 2);
ipj_set_value(&iri_device, E_IPJ_KEY_READ_WORD_COUNT, 6);
```

After the Indy SiP is configured:

- Start inventory using *ipj_start*
- Process *Tag Operation Reports* to confirm success

Retry Mechanism The Access Retry feature is enabled using the *E_IPJ_KEY_TAG_OPERATION_RETRIES* key. The Access operation is retried up to maximum number of retries specified by the key. A retry is not attempted if the Tag returns a Memory Overrun or Memory Locked error condition. The number of attempted retries is reported in the *Tag Operation Report*

GPIO Configuration

The Indy SiP possesses 4 user controllable GPIO pins which can be configured as general purpose outputs, inputs, or action-attached inputs. Outputs can be driven either high or low. Inputs exist in either a high-impedance, pulled up, or pulled down state.

Note: The Default state for all GPIO pins is high-impedance input.

The following keys enable the GPIO operation:

- *E_IPJ_KEY_GPIO_MODE*
- *E_IPJ_KEY_GPIO_STATE*
- *E_IPJ_KEY_GPIO_HI_ACTION*
- *E_IPJ_KEY_GPIO_LO_ACTION*
- *E_IPJ_KEY_GPIO_DEBOUNCE_MS*

Note: GPIO_PULSE is not yet supported

Each key is banked and therefore requires the use of an index value corresponding to the appropriate GPIO (1-4). Index 0 is reserved for future functionality.

Once all appropriate keys are set, the GPIO configuration is initialized by sending the Indy SiP a *ipj_start* command with the *action* argument *E_IPJ_ACTION_GPIO*.

GPIO Output

To configure GPIO 1 as an output and set it's state to logic high (+3v3):

```
ipj_set(&iri_device, E_IPJ_KEY_GPIO_MODE, 1, 0, E_IPJ_GPIO_MODE_OUTPUT);
ipj_set(&iri_device, E_IPJ_KEY_GPIO_STATE, 1, 0, E_IPJ_GPIO_STATE_HI);
ipj_start(&iri_device, E_IPJ_ACTION_GPIO);
ipj_stop(&iri_device, E_IPJ_ACTION_GPIO);
```

GPIO Input

To configure GPIO 1 as an input with an internal pull-down (0v):

```
ipj_set(&iri_device, E_IPJ_KEY_GPIO_MODE, 1, 0, E_IPJ_GPIO_MODE_INPUT);
ipj_set(&iri_device, E_IPJ_KEY_GPIO_STATE, 1, 0, E_IPJ_GPIO_STATE_HI);
ipj_start(&iri_device, E_IPJ_ACTION_GPIO);
/* At this point, the user should call ipj_receive and handle any
 * incoming GPIO reports. When the desired amount of reports have
 * been collected, the user should call ipj_stop and poll for the gpio
 * stop report */
ipj_stop(&iri_device, E_IPJ_ACTION_GPIO);
```

GPIO Input w/ Action

To configure GPIO 1 as a floating input that will start inventory when it is pulled high (+3v3):

```
ipj_set(&iri_device, E_IPJ_KEY_GPIO_MODE, 1, 0, E_IPJ_GPIO_MODE_INPUT_ACTION);
ipj_set(&iri_device, E_IPJ_KEY_GPIO_STATE, 1, 0, E_IPJ_GPIO_STATE_FLOAT);
ipj_set(&iri_device, E_IPJ_KEY_GPIO_HI_ACTION, 1, 0, E_IPJ_GPI_ACTION_START_INVENTORY);
ipj_start(&iri_device, E_IPJ_ACTION_GPIO);
/* At this point, the user should begin calling ipj_receive(). When
 * there is a logic-high event on the selected pin, the user will
 * receive a GPIO report, followed by tag reports as tags come into
 * the field. When the user has received the desired number of tag
 * reports, they should:
 * 1. issue an ipj_stop(&iri_device, E_IPJ_ACTION_INVENTORY) command
 * 2. call ipj_receive until they receive an inventory stop report.
 * 3. issue an ipj_stop(&iri_device, E_IPJ_ACTION_GPIO) command
 * 4. call ipj_receive until they receive a gpio stop report
 */
ipj_stop(&iri_device, E_IPJ_ACTION_GPIO);
```

It is currently only possible to start and stop inventory via an input pin. Please see [ipj_gpi_action](#) for more info

GPIO Input Debounce

The GPIO input logic allows for a user programmable debounce period to be set in increments of 1 ms. If the debounce period is set to 0 (default), any input events will be reported and (if appropriately configured) acted upon. If it is set to > 0, the firmware will suppress any events that fall short of fulfilling this lockout period.

Furthermore, the internal GPI control is handled via polling, and the debounce timer has a tickrate of 100 us. As a result, the best-case response time will be `debounce_time +/- 100 us`. The user should therefore not set the debounce time too close to the total expected period of the input trigger signal.

GPIO Input Debounce Example Debounce period is set to 9 ms. The target GPI is held high for *exactly* 9 ms. Due to the resolution of the debounce timer, the debounce clock expires at 9.09 ms, and the event is missed.

For the Above scenario to function as intended, either the debounce rate would need to be reduced to 8 ms, or the GPI hold time would need to be extended.

Setting the debounce key:

```
/* Set the Debounce rate for GPIO 1 to 8ms */
ipj_set(&iri_device, E_IPJ_KEY_GPIO_DEBOUNCE_MS, 1, 0, 8);
```

Test Commands

Transmit CW

For evaluation purposes, the Indy SiP can be configured to transmit CW on a fixed frequency.

Fixed Frequency The following code snippet sets the test frequency to 912.25 MHz:

```
ipj_set_value(&iri_device, E_IPJ_KEY_TEST_ID, E_IPJ_TEST_ID_SET_FREQUENCY);
ipj_set_value(&iri_device, E_IPJ_KEY_TEST_PARAMETERS, 912250);
ipj_set_value(&iri_device, E_IPJ_KEY_TEST_PARAMETERS, 0);
```

```

ipj_stopped_flag = 0;
ipj_start(&iri_device, E_IPJ_ACTION_TEST);
while (!ipj_stopped_flag)
{
    ipj_receive(&iri_device); /* Report handler to set ipj_stopped_flag=1 on stop_report */
}

```

Turn CW On The following code snippet turns on CW:

```

ipj_set_value(&iri_device, E_IPJ_KEY_TEST_ID, E_IPJ_TEST_ID_CW_CONTROL);
ipj_set_value(&iri_device, E_IPJ_KEY_TEST_PARAMETERS, 1);
ipj_stopped_flag = 0;
ipj_start(&iri_device, E_IPJ_ACTION_TEST);
while (!ipj_stopped_flag)
{
    ipj_receive(&iri_device); /* Report handler to set ipj_stopped_flag=1 on stop_report */
}

```

Turn CW Off The following code snippet turns off CW:

```

ipj_set_value(&iri_device, E_IPJ_KEY_TEST_ID, E_IPJ_TEST_ID_CW_CONTROL);
ipj_set_value(&iri_device, E_IPJ_KEY_TEST_PARAMETERS, 0);
ipj_stopped_flag = 0;
ipj_start(&iri_device, E_IPJ_ACTION_TEST);
while (!ipj_stopped_flag)
{
    ipj_receive(&iri_device); /* Report handler to set ipj_stopped_flag=1 on stop_report */
}

```

Transmit PRBS

For evaluation purposes, the Indy SiP can be configured to transmit random data (PRBS) on a fixed frequency.

Fixed Frequency The following code snippet sets test frequency to 912.25 MHz:

```

ipj_set_value(&iri_device, E_IPJ_KEY_TEST_ID, E_IPJ_TEST_ID_SET_FREQUENCY);
ipj_set_value(&iri_device, E_IPJ_KEY_TEST_PARAMETERS, 912250);
ipj_stopped_flag = 0;
ipj_start(&iri_device, E_IPJ_ACTION_TEST);
while (!ipj_stopped_flag)
{
    ipj_receive(&iri_device); /* Report handler to set ipj_stopped_flag=1 on stop_report */
}

```

Turn PRBS On The following code snippet turns on random data:

```

ipj_set_value(&iri_device, E_IPJ_KEY_TEST_ID, E_IPJ_TEST_ID_PRBS_CONTROL);
ipj_set_value(&iri_device, E_IPJ_KEY_TEST_PARAMETERS, 1);
ipj_stopped_flag = 0;
ipj_start(&iri_device, E_IPJ_ACTION_TEST);
while (!ipj_stopped_flag)
{

```

```

    ipj_receive(&iri_device); /* Report handler to set ipj_stopped_flag=1 on stop_report */
}

```

Turn PRBS Off The following code snippet turns off random data:

```

ipj_set_value(&iri_device, E_IPJ_KEY_TEST_ID, E_IPJ_TEST_ID_PRBS_CONTROL);
ipj_set_value(&iri_device, E_IPJ_KEY_TEST_PARAMETERS, 0);
ipj_stopped_flag = 0;
ipj_start(&iri_device, E_IPJ_ACTION_TEST);
while (!ipj_stopped_flag)
{
    ipj_receive(&iri_device); /* Report handler to set ipj_stopped_flag=1 on stop_report */
}

```

Get Temperature

The following code snippet gets the current internal and external temperature. The test report includes the same temperature data as the `...TEST_RESULT` keys. The temperature values represent degrees C. The values represented by each of the test result keys are shown in the table below.

Table 1.6: Temperature test results

Key	Value represented
E_IPJ_KEY_TEST_RESULT_1	SiP internal controller temperature
E_IPJ_KEY_TEST_RESULT_2	SiP external estimated temperature
E_IPJ_KEY_TEST_RESULT_3	SiP PA temperature (RS2000 only)

```

uint32_t internal_temperature;
uint32_t external_temperature;
uint32_t pa_temperature;

ipj_set_value(&iri_device, E_IPJ_KEY_TEST_ID, E_IPJ_TEST_ID_TEMPERATURE_CONTROL);
ipj_set_value(&iri_device, E_IPJ_KEY_TEST_PARAMETERS, 1);
ipj_stopped_flag = 0;
ipj_start(&iri_device, E_IPJ_ACTION_TEST);
while (!ipj_stopped_flag)
{
    ipj_receive(&iri_device); /* Report handler to set ipj_stopped_flag=1 on stop_report */
}
error = ipj_get_value(&iri_device, E_IPJ_KEY_TEST_RESULT_1, &internal_temperature);
error = ipj_get_value(&iri_device, E_IPJ_KEY_TEST_RESULT_2, &external_temperature);
error = ipj_get_value(&iri_device, E_IPJ_KEY_TEST_RESULT_3, &pa_temperature);

```

Custom Configuration

Report Field Configuration

The Indy SiPs generate Tag Operation Reports when tags are inventoried. Tag Operation Reports include the `ipj_tag_operation_report` structure. The `ipj_tag_operation_report` structure includes the `ipj_tag` structure. The fields within the `ipj_tag` structure are optional. Only the EPC, TID (when FastId is enabled) and Timestamp fields are enabled by default.

Configure report fields using the following key:

- *E_IPJ_KEY_REPORT_CONTROL_TAG*

Example report configuration:

```
ipj_set_value(
    &iri_device,
    E_IPJ_KEY_REPORT_CONTROL_TAG,
    E_IPJ_TAG_FLAG_BIT_EPC          | E_IPJ_TAG_FLAG_BIT_TID |
    E_IPJ_TAG_FLAG_BIT_TIMESTAMP | E_IPJ_TAG_FLAG_BIT_CHANNEL);
```

Custom Regulatory Region

The Indy SiPs support a broad range of regulatory regions. The regulatory regions are listed [here](#).

Users can create a custom region by configuring an Indy SiP appropriately. Configure an Indy SiP for custom region using the following keys:

- *E_IPJ_KEY_REGION_ON_TIME_NOMINAL*
- *E_IPJ_KEY_REGION_ON_TIME_ACCESS*
- *E_IPJ_KEY_REGION_OFF_TIME*
- *E_IPJ_KEY_REGION_OFF_TIME_SAME_CHANNEL*
- *E_IPJ_KEY_REGION_START_FREQUENCY_KHZ*
- *E_IPJ_KEY_REGION_CHANNEL_SPACING_KHZ*
- *E_IPJ_KEY_REGION_RANDOM_HOP*
- *E_IPJ_KEY_REGION_INDY_PLL_R_DIVIDER*
- *E_IPJ_KEY_REGION_RF_FILTER*
- *E_IPJ_KEY_REGION_CHANNEL_TABLE_SIZE*
- *E_IPJ_KEY_REGION_CHANNEL_TABLE*

Change the region to *E_IPJ_REGION_CUSTOM* after configuring the first 8 keys and before configuring the channel table and channel table size. Custom region settings are applied when the *E_IPJ_KEY_REGION_ID* is set for *E_IPJ_REGION_CUSTOM*. If subsequent changes are required, change the region to another region and set back to *E_IPJ_REGION_CUSTOM* to apply the settings.

E_IPJ_KEY_REGION_INDY_PLL_R_DIVIDER controls the frequency resolution of the RF carrier. RF carrier frequency equals $(6*N/R)$ MHz, where N is an integer. The RF carrier frequency is limited to a given range for each Indy SiP SKU. Set the Indy PLL R Divider according the following table.

Table 1.7: Indy Pll R Divider

Di-vider	Frequency Resolution (or Frequency Step Size)	Example Region
24	250 kHz	E_IPJ_REGION_FCC_PART_15_247
30	200 kHz	E_IPJ_REGION_SOUTH_AFRICA_915_919_MHZ
48	125 kHz	E_IPJ_REGION_CHINA_920_925_MHZ
60	100 kHz	E_IPJ_REGION_ETSI_EN_302_208_V1_4_1

Note: Custom region keys are only applied after the *E_IPJ_KEY_REGION_ID* is changed to *E_IPJ_REGION_CUSTOM*.

Example FCC configuration:

```

ipj_error setup_custom_region_fcc(ipj_iri_device* iri_device)
{
    unsigned int i;
    ipj_error error;
    ipj_key_value key_value[16];
    uint32_t key_value_count=0;
    ipj_key_list key_list;
    uint32_t key_list_count=0;

    /*
     * 1. Setup Custom Region
     */
    key_value_count = 9;

    memset(key_value, 0, sizeof(key_value));
    memset(&key_list, 0, sizeof(key_list));

    key_value[0].key = E_IPJ_KEY_REGION_ON_TIME_NOMINAL;
    key_value[0].value = 200;

    key_value[1].key = E_IPJ_KEY_REGION_ON_TIME_ACCESS;
    key_value[1].value = 400;

    key_value[2].key = E_IPJ_KEY_REGION_OFF_TIME;
    key_value[2].value = 0;

    key_value[3].key = E_IPJ_KEY_REGION_OFF_TIME_SAME_CHANNEL;
    key_value[3].value = 0;

    key_value[4].key = E_IPJ_KEY_REGION_START_FREQUENCY_KHZ;
    key_value[4].value = 902750;

    key_value[5].key = E_IPJ_KEY_REGION_CHANNEL_SPACING_KHZ;
    key_value[5].value = 500;

    key_value[6].key = E_IPJ_KEY_REGION_RANDOM_HOP;
    key_value[6].value = 1;

    key_value[7].key = E_IPJ_KEY_REGION_INDY_PLL_R_DIVIDER;
    key_value[7].value = 24;

    key_value[8].key = E_IPJ_KEY_REGION_RF_FILTER;
    key_value[8].value = 0;

    error = ipj_bulk_set(iri_device, &key_value[0], key_value_count, NULL, 0);
    if (error)
    {
        printf("ERROR: IPJ_BULK_SET FAILED - ERROR CODE: %d\n\n", error);
        return error;
    }

    /*
     * 2. Change Region
     * Note: Custom region params are only applied once and cannot be updated on the fly. Further
     * they are only applied if the region changes. Force the region to something non-custom,
     * then change to custom so the settings are applied.
     */
    error = ipj_set_value(iri_device, E_IPJ_KEY_REGION_ID, E_IPJ_REGION_FCC_PART_15_247);
}

```

```

if (error)
{
    printf("ERROR: Set E_IPJ_KEY_REGION_ID FAILED - ERROR CODE: %d\n\n", error);
    return error;
}
error = ipj_set_value(iri_device, E_IPJ_KEY_REGION_ID, E_IPJ_REGION_CUSTOM);
if (error)
{
    printf("ERROR: Set E_IPJ_KEY_REGION_ID FAILED - ERROR CODE: %d\n\n", error);
    return error;
}

/*
 * 3. Setup Channel Table
 */
key_list_count = 1;
key_list.key = E_IPJ_KEY_REGION_CHANNEL_TABLE;
key_list.list_count = 32; /* Max size for key_list */
key_list.has_value_index = true;
key_list.value_index = 0;

/* Set the first part of the channel table*/
for (i = 0; i < key_list.list_count; i++)
{
    /* Channels are one based */
    key_list.list[i] = i + 1;
}

error = ipj_bulk_set(iri_device, NULL, 0, &key_list, key_list_count);
if (error)
{
    printf("ERROR: IPJ_BULK_SET FAILED - ERROR CODE: %d\n\n", error);
    return error;
}

/* The rest of the channels */
key_list.list_count = 18;
key_list.has_value_index = true;
key_list.value_index = 32;

/* Set the first part of the channel table*/
for (i = 0; i < key_list.list_count; i++)
{
    /* Channels are one based, pick up where we left off */
    key_list.list[i] = i + 32 + 1;
}

error = ipj_bulk_set(iri_device, NULL, 0, &key_list, key_list_count);
if (error)
{
    printf("ERROR: IPJ_BULK_SET FAILED - ERROR CODE: %d\n\n", error);
    return error;
}

error = ipj_set_value(iri_device, E_IPJ_KEY_REGION_CHANNEL_TABLE_SIZE, 50);
if (error)
{
    printf("ERROR: Set E_IPJ_KEY_REGION_CHANNEL_TABLE_SIZE FAILED - ERROR CODE: %d\n\n", error);
}

```

```

        return error;
    }

    return E_IPJ_ERROR_SUCCESS;
}

```

Example ETSI configuration:

```

ipj_error setup_custom_region_etsi(ipj_iri_device* iri_device)
{
    ipj_error error;
    ipj_key_value    key_value[16];
    uint32_t         key_value_count=0;
    ipj_key_list     key_list;
    uint32_t         key_list_count=0;

    /*
     * 1. Setup Custom Region
     */
    key_value_count = 8;

    memset(key_value, 0, sizeof(key_value));
    memset(&key_list, 0, sizeof(key_list));

    key_value[0].key    = E_IPJ_KEY_REGION_ON_TIME_NOMINAL;
    key_value[0].value = 3800;

    key_value[1].key    = E_IPJ_KEY_REGION_ON_TIME_ACCESS;
    key_value[1].value = 4000;

    key_value[2].key    = E_IPJ_KEY_REGION_OFF_TIME;
    key_value[2].value = 0;

    key_value[3].key    = E_IPJ_KEY_REGION_OFF_TIME_SAME_CHANNEL;
    key_value[3].value = 100;

    key_value[4].key    = E_IPJ_KEY_REGION_START_FREQUENCY_KHZ;
    key_value[4].value = 865100;

    key_value[5].key    = E_IPJ_KEY_REGION_CHANNEL_SPACING_KHZ;
    key_value[5].value = 200;

    key_value[6].key    = E_IPJ_KEY_REGION_RANDOM_HOP;
    key_value[6].value = 0;

    key_value[7].key    = E_IPJ_KEY_REGION_INDY_PLL_R_DIVIDER;
    key_value[7].value = 60;

    error = ipj_bulk_set(iri_device, &key_value[0], key_value_count, NULL, 0);
    if (error)
    {
        printf("ERROR: IPJ_BULK_SET FAILED - ERROR CODE: %d\n\n", error);
        return error;
    }

    /*
     * 2. Change Region
     * Note: Custom region params are only applied once and cannot be updated on the fly. Further
     * they are only applied if the region changes. Force the region to something non-custom,

```

```

    * then change to custom so the settings are applied.
    */
error = ipj_set_value(iri_device, E_IPJ_KEY_REGION_ID, E_IPJ_REGION_FCC_PART_15_247);
if (error)
{
    printf("ERROR: Set E_IPJ_KEY_REGION_ID FAILED - ERROR CODE: %d\n\n", error);
    return error;
}
error = ipj_set_value(iri_device, E_IPJ_KEY_REGION_ID, E_IPJ_REGION_CUSTOM);
if (error)
{
    printf("ERROR: Set E_IPJ_KEY_REGION_ID FAILED - ERROR CODE: %d\n\n", error);
    return error;
}

/*
 * 3. Setup Channel Table
 */
key_list_count = 1;
key_list.key = E_IPJ_KEY_REGION_CHANNEL_TABLE;
key_list.list_count = 4;

key_list.list[0] = 4;
key_list.list[1] = 7;
key_list.list[2] = 10;
key_list.list[3] = 13;

error = ipj_bulk_set(iri_device, NULL, 0, &key_list, key_list_count);
if (error)
{
    printf("ERROR: IPJ_BULK_SET FAILED - ERROR CODE: %d\n\n", error);
    return error;
}

error = ipj_set_value(iri_device, E_IPJ_KEY_REGION_CHANNEL_TABLE_SIZE, key_list.list_count);
if (error)
{
    printf("ERROR: Set E_IPJ_KEY_REGION_CHANNEL_TABLE_SIZE FAILED - ERROR CODE: %d\n\n", error);
    return error;
}

return E_IPJ_ERROR_SUCCESS;
}

```

Change Baud Rate

The Indy SiP serial interface operates at 115,200 Baud unless configured otherwise, either by setting the key at run-time or using the *Stored Settings* functionality.

The *ipj_modify_connection* function allows the user to change the UART Baud rate.

The following code snippet changes serial interface to 57,600 Baud:

```

ipj_connection_params connection_params;
connection_params.serial.baudrate = E_IPJ_BAUD_RATE_BR57600;
error = ipj_modify_connection(&iri_device, E_IPJ_CONNECTION_TYPE_SERIAL, &connection_params);

```

Application Update

Updates to the application portion of the firmware (also known as bootloads) are handled via the IRI Protocol using a specially prepared binary image.

There is a full example of how to update the Indy SiP application image in the IRI example *IRI Loader Example*.

Note: If the wrong image file is used for application update, the firmware onboard the SiP can be damaged, possibly preventing future updates. Care should be taken when updating to select the correct image.

The format of the image is as follows:

Chunk Size (4 bytes)
Erase Block (chunk_size)
Load Block 1
Load Block 2
...
Load Block n

Example to update the device:

```
int chunk_size;
FILE* image_file_handle;
uint8_t file_buf[256];

/* Put the device in recovery mode */
error = ipj_reset(&iri_device, E_IPJ_RESET_TYPE_TO_BOOTLOADER);
if(error)
{
    return error;
}

image_file_handle = fopen("<path_to_image>", "rb");

if(image_file_handle == NULL)
{
    return -1;
}

/* Get the image chunk size. This is stored in the first 32 bits
 * of the upgrade image */
if(fread(file_buf, 4, 1, image_file_handle) == 0)
{
    return -1;
}

chunk_size = (file_buf[0] & 0xff) | (file_buf[1] << 8) |
(file_buf[2] << 16) | (file_buf[3] << 24);

if(chunk_size < 22 || chunk_size > 270)
{
    return -1;
}

/* For each chunk in the image file, write it to the RS500 */
while(fread(file_buf, chunk_size, 1, image_file_handle) > 0)
{
```

```

    error = ipj_flash_handle_loader_block(&iri_device, chunk_size, file_buf);
    if (error)
    {
        return error;
    }
}

/* Reset the device to resume operation */
error = ipj_reset(&iri_device, E_IPJ_RESET_TYPE_SOFT);
if(error)
{
    printf("Unable to reset\n");
    return error;
}
}

```

Note: This example demonstrates how to update the image from a host using POSIX functions like fread. Your method for reading/streaming the data may vary.

Error Handling

The Indy SiP error handling scheme consists of 2 components:

- Error Keys
- Error Reports

Error Keys are set any time an error occurs.

There are 3 user readable keys:

- *E_IPJ_KEY_FIRST_ERROR* - Set the first time an error is detected
- *E_IPJ_KEY_LAST_ERROR* - Updated any time an error is detected
- *E_IPJ_KEY_SYSTEM_ERROR* - Set after critical system errors which cause a reboot

All of these keys can be read at any time and contain the error ID as well as up to 4 parameters detailing the specifics of the error (these parameters can be found in the error codes documentation). Additionally, First Error can be cleared by issuing an *ipj_start* command with a CLEAR_ERROR action.

Note: Only first error can be user cleared. Last and System errors can only be cleared by a power cycle.

Example - To Read the First Error

```

ipj_key_list key_list;
memset(&key_list, 0, sizeof(key_list));
key_list.key = E_IPJ_KEY_FIRST_ERROR;
key_list.list_count = 5;
error = ipj_bulk_get(&iri_device, NULL, 0, &key_list, 1);
if (error)
{
    printf("ERROR: IPJ_BULK_GET FAILED - ERROR CODE: %d\n\n", error);
    return error;
}

```

Example - To Clear the Error

```
error = ipj_start(&iri_device, E_IPJ_ACTION_CLEAR_ERROR);
if (error)
{
    printf("ERROR: IPJ_START FAILED - ERROR CODE: %d\n\n", error);
    return error;
}
```

Note: The device will automatically send a stop report once the error is clear, no other action is necessary. Attempting to send *ipj_stop* with a CLEAR_ERROR action will result in an error.

Error Reports are generated any time an error is detected and the host happens to be listening for incoming data (i.e. after a start inventory command has been issued). An error report contains the same information as an error key: Error ID + up to 4 parameters.

Note: Error reports will only be generated once per error to avoid the host from being flooded with reports should the same error occur multiple times in rapid succession.

Power Management

The Indy SiP has 5 operating states:

- Active - RF is on and Indy SiP is actively communicating with tags
- Low Latency Idle - RF is partially on and Indy SiP is awaiting events/instruction
- Standard Idle - RF is off and Indy SiP is awaiting events/instruction
- Standby - RF is off, all operations are halted, and all configuration state is conserved. Indy SiP awaits USART traffic or a wakeup pin event.
- Sleep - RF is off, all operations are stopped, and all configuration state is lost. Indy SiP awaits a wakeup pin or reset pin event.

There are two idle modes, Low Latency and Standard.

- Standard Idle - In standard idle mode, the Indy SiP will suspend all activity except for the Host UART.
- Low Latency Idle - In low latency idle mode, the Indy SiP will suspend all activity except for the Host UART and partially enabled RF function configuration. This mode allows for a faster RFID command response time versus the standard mode at the cost of a high idle current. The latency time savings is approximately 8ms.

Changing the idle mode:

```
/* Sets the idle power mode to standard. This automatically takes effect */
ipj_set_value(&iri_device, E_IPJ_KEY_DEVICE_IDLE_POWER_MODE,
              E_IPJ_IDLE_POWER_MODE_STANDARD);

/* Sets the idle power mode to low-latency. This automatically takes effect */
ipj_set_value(&iri_device, E_IPJ_KEY_DEVICE_IDLE_POWER_MODE,
              E_IPJ_IDLE_POWER_MODE_LOW_LATENCY);
```

There are two low power modes, Standby and Sleep.

- Standby - In standby mode, the Indy SiP suspends all activity and waits in a low power mode until the host application wakes it up. Upon being awoken, all operations resume as they were (state has been saved). Wake up occurs when the host application toggles the WKUP pin, or a GPIO event on a pin configured as an input / input+action is detected. It is also possible to wake up the Indy SiP via USART traffic, however some data will be lost in the transmission, so it is necessary to retry until the device successfully responds (we recommend performing an `ipj_get_value` until successful or a timeout / desired retry count has elapsed)
- Sleep - In sleep, the device holds itself in reset until either the WKUP pin is pulled high, or the reset line is toggled. No state is saved and this can be considered to be equivalent to asserting and holding the reset line.

Putting the device into standby/sleep:

```

/* Put the device into standby mode */
error = ipj_start(&iri_device, E_IPJ_ACTION_STANDBY);
if (error)
{
    printf("ERROR: STANDBY FAILED - ERROR CODE: %x\n\n", error);
    return error;
}

printf("Attempting to wake RS500\n");

/* Set the receive timeout MS to a lower number (100ms) */
error = ipj_set_receive_timeout_ms(&iri_device, 100);
if (error)
{
    printf("ERROR: SET RX TIMEOUT FAILED - ERROR CODE: %x\n\n", error);
    return error;
}

/* Spin on IPJ_GET_VALUE (any key will do) until the device responds */
do
{
    error = ipj_get_value(&iri_device, E_IPJ_KEY_APPLICATION_VERSION, &value);
}
while (error && retries--);
if (error)
{
    printf("ERROR: Failed to wake RS500 device - ERROR CODE: %x\n\n", error);
    return error;
}

/* At this point, the device is active and can perform RFID operations */

/* Now put the device in sleep mode */
error = ipj_start(&iri_device, E_IPJ_ACTION_SLEEP);
if (error)
{
    printf("ERROR: STANDBY FAILED - ERROR CODE: %x\n\n", error);
    return error;
}

/* The host application must toggle either the RESET or WKUP pins to
 * bring the device out of sleep */

```

Get Info

It is possible to use IRI to query the Indy SiP about the keys it contains.

The *ipj_get_info* function allows the user to detect the size of the data that the key contains (uint8/uint32/etc) as well as the number of banks (count), the number of values (length), and any read/write permissions that may be set on the key.

Please note that the information contained in the *ipj_key_info* struct refers to maximums, i.e. just because a key claims to have a length of 50 doesn't necessarily mean that all 50 values are used.

The following code snippet gets the *ipj_key_info* for the *E_IPJ_KEY_REGION_CHANNEL_TABLE* key and prints out the value contained within for each possible *value_index*:

```
ipj_key_info keyinfo;
uint32_t i;
uint32_t result;

IPJ_CLEAR_STRUCT(keyinfo);
/* Get keyinfo for region channel table */
error = ipj_get_info(&iri_device, E_IPJ_KEY_REGION_CHANNEL_TABLE, &keyinfo);
if (error)
{
    printf("Error getting channel table. Error: %x\n\n", error);
    return error;
}

/* Loop through the channel table and print the contents */
for(i = 0; i < keyinfo.length; i++)
{
    error = ipj_get(&iri_device, E_IPJ_KEY_REGION_CHANNEL_TABLE,
                  0, /* Bank_index of 0 since only one exists */
                  i, /* Value_index of i as we're looping through all */
                  &result);

    if (error)
    {
        printf("Error getting channel value. Error: %x\n\n", error);
    }

    printf("Found Channel %d at value_index %d\n", result, i);
}
```

1.5.8 Functions

This section describes API functions to communicate with the Impinj Indy Reader SiP.

Device Management

ipj_get_api_version

uint32_t *ipj_get_api_version*()

This function returns the API version.

Return

uint32_t

`ipj_initialize_iri_device`

`ipj_error ipj_initialize_iri_device(ipj_iri_device * iri_device)`

This function initializes the IRI device data structure.

Parameters

- `iri_device` - IRI device data structure

Before using an IRI device, the User application must do the following: allocate or declare IRI device data structure, initialize IRI device data structure (this function), register platform handlers, and connect to the IRI device.

Return

`ipj_error`

`ipj_deinitialize_iri_device`

`ipj_error ipj_deinitialize_iri_device(ipj_iri_device * iri_device)`

This function deinitializes the IRI device data structure.

Parameters

- `iri_device` - IRI device data structure

User may deinitialize IRI device after disconnecting from the IRI device. Once the IRI device is deinitialized, the User may de-allocate the IRI device data structure.

Return

`ipj_error`

`ipj_register_handler`

`ipj_error ipj_register_handler(ipj_iri_device * iri_device, ipj_handler_type handler_type, IPJ_VOID_PFN handler)`

This function registers platform and report handlers.

Parameters

- `iri_device` - IRI device data structure
- `handler_type` - Type of handler to register
- `handler` - Handler pointer

Before using an IRI device, the User application must do the following: allocate or declare IRI device data structure, initialize IRI device data structure, register platform handlers (this function or the `_ext`-suffixed version), and connect to the IRI device.

User application registers the following platform handlers: platform open port handler, platform close port handler, platform receive handler, platform transmit handler, and platform timestamp handler.

User application registers report handler to handle the following reports: tag operation report, error report, and stop report

Return

ipj_error

ipj_connect

ipj_error **ipj_connect**(*ipj_iri_device* * iri_device, *IPJ_READER_IDENTIFIER* reader_identifier, ipj_connection_type connection_type, ipj_connection_params * params)

This function opens the serial port and connects the Host to the IRI device.

Parameters

- *iri_device* - IRI device data structure
- *reader_identifier* - Identifier associated with IRI device
- *connection_type* - Type of connection (Serial/TCP/etc)
- *params* - Connection parameters (baudrate/etc) (NULL==default)

Before using an IRI device, the User application must do the following: allocate or declare IRI device data structure, initialize IRI device data structure, register platform handlers, and connect to the IRI device (this function).

Return

ipj_error

ipj_disconnect

ipj_error **ipj_disconnect**(*ipj_iri_device* * iri_device)

This function disconnects the Host from the IRI device and closes the serial port.

Parameters

- *iri_device* - IRI device data structure

User may disconnect from the IRI device after all operations are completed.

Return

ipj_error

ipj_modify_connection

ipj_error **ipj_modify_connection**(*ipj_iri_device* * iri_device, ipj_connection_type connection_type, ipj_connection_params * params)

This function modifies the serial port baud rate used to talk to an IRI device.

Parameters

- *iri_device* - IRI device data structure

- `connection_type` - Connection modification type
- `params` - Connection modification parameters

Before modifying an iri connection, the user must have already successfully connected to a device.

Return

`ipj_error`

`ipj_suppress_set_responses`

`ipj_error ipj_suppress_set_responses(ipj_iri_device * iri_device)`

This function allows the user to suppress all ‘set’ and ‘bulk-set’ action responses generated by the reader.

The user will be responsible for checking and dealing with errors on the device.

Return

`ipj_error`

`ipj_resume_set_responses`

`ipj_error ipj_resume_set_responses(ipj_iri_device * iri_device)`

This function allows the user to resume generation of ‘set’ action responses by the reader.

All ‘set’ and ‘bulk-set’ actions will receive an associated response containing success/failure status of the command (This is the default)

Return

`ipj_error`

`ipj_set_receive_timeout_ms`

`ipj_error ipj_set_receive_timeout_ms(ipj_iri_device * iri_device, uint32_t timeout_ms)`

This function allows the user to set the amount of time the receive state machine should wait before returning a timeout error.

Return

`ipj_error`

Operational Commands

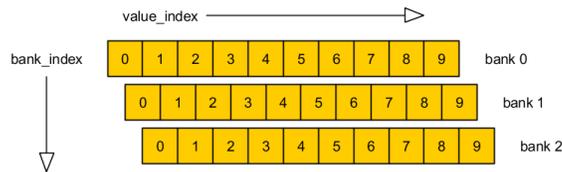
Note: `ipj_set/ipj_get` and `ipj_bulk_set/ipj_bulk_get` require the inclusion of `bank_index` and `value_index` parameters. These parameters correspond to the position of the value in the key that the user is trying to modify.

The majority of keys are a single value/bank which require a `bank_index` and `value_index` of 0. In these cases, we recommend the user use the convenience functions `ipj_set_value` and `ipj_get_value` (which masks this detail).

There are also three types of ‘complex’ keys:

- Key lists - These are individual keys which can contain a list of data. An example of this would be channel tables. In these cases, the user would supply a `value_index` corresponding to the particular item in the list they would like to access. (These are indexed starting at 0).
- Banked Keys - These are keys which exist in several ‘banks’ or instances. An example of this would be the `SELECT_XXX` keys. To access the settings for a select command instance, you would provide its bank number. (These are indexed starting at 0).
- Banked Key lists - These are keys which contain lists of values, and which also exist as several ‘banks’ or instances. An example of this would be the `E_IPJ_KEY_SELECT_MASK_VALUE` Key. The `value_index` would correspond to the mask values for the select command instance, and the `bank_index` corresponds to the particular select command instance.

The following diagram details this layout:



`ipj_reset`

`ipj_error ipj_reset(ipj_iri_device * iri_device, ipj_reset_type reset_type)`

This function resets the IRI device.

Parameters

- `iri_device` - IRI device data structure
- `reset_type` - Type of reset to perform

State of the IRI device is not preserved across resets.

Return

`ipj_error`

`ipj_set`

`ipj_error ipj_set(ipj_iri_device * iri_device, ipj_key key, uint32_t bank_index, uint32_t value_index, uint32_t value)`

This function sets the value of the specified key code on the IRI device.

Parameters

- `iri_device` - IRI device data structure
- `key` - Key code to set
- `bank_index` - Key code bank_index to set (applicable for key codes with more than one copy)
- `value_index` - Key code value_index to set (applicable for key codes with lists)

- `value` - Value to set

Return

`ipj_error`

`ipj_set_value`

Note: This function is a simplified version of `ipj_set` which assumes a `bank_index` and `value_index` of 0

`ipj_error ipj_set_value(ipj_iri_device * iri_device, ipj_key key, uint32_t value)`

This function sets the value of the specified key code on the IRI device.

Parameters

- `iri_device` - IRI device data structure
- `key` - Key code to set
- `value` - Value to set

Return

`ipj_error`

`ipj_get`

`ipj_error ipj_get(ipj_iri_device * iri_device, ipj_key key, uint32_t bank_index, uint32_t value_index, uint32_t * value)`

This function retrieves the value of the specified key code from the IRI device.

Parameters

- `iri_device` - IRI device data structure
- `key` - Key code to get
- `bank_index` - Key code `bank_index` to get (applicable if more than one copy of a key)
- `value_index` - Key code `value_index` to get (applicable for key code with lists)
- `value` - Data buffer to store retrieved value

Return

`ipj_error`

`ipj_get_value`

Note: This is a simplified version of `ipj_get` which assumes a `bank_index` and `value_index` of 0

`ipj_error ipj_get_value(ipj_iri_device * iri_device, ipj_key key, uint32_t * value)`

This function retrieves the value of the specified key code from the IRI device.

Parameters

- `iri_device` - IRI device data structure
- `key` - Key code to get
- `value` - Data buffer to store retrieved value

Return

`ipj_error`

`ipj_bulk_set`

`ipj_error ipj_bulk_set(ipj_iri_device * iri_device, ipj_key_value * key_value, uint32_t key_value_count, ipj_key_list * key_list, uint32_t key_list_count)`

This function sets the value of the specified key codes on the IRI device.

Parameters

- `iri_device` - IRI device data structure
- `key_value` - Array of key codes and values to set
- `key_value_count` - Number of key codes and values to set
- `key_list` - Array of key codes to set (used for key codes with lists)
- `key_list_count` - Number of keys codes (with lists) to set

Return

`ipj_error`

`ipj_bulk_get`

`ipj_error ipj_bulk_get(ipj_iri_device * iri_device, ipj_key_value * key_value, uint32_t key_value_count, ipj_key_list * key_list, uint32_t key_list_count)`

This function retrieves the values of the specified key codes from the IRI device.

Parameters

- `iri_device` - IRI device data structure
- `key_value` - Array of key codes to retrieve
- `key_value_count` - Number of key codes and values to retrieve
- `key_list` - Array of key codes to retrieve (used for key codes with lists)
- `key_list_count` - Number of keys codes (with lists) to retrieve

Return

`ipj_error`

`ipj_get_info`

`ipj_error ipj_get_info(ipj_iri_device * iri_device, ipj_key key, ipj_key_info * key_info)`

This function retrieves the value of the specified key code from the IRI device.

Parameters

- `iri_device` - IRI device data structure
- `key` - Key code to get info
- `key_info` - Information about the specified key code

Return

`ipj_error`

`ipj_start`

`ipj_error ipj_start(ipj_iri_device * iri_device, ipj_action action)`

This function starts the specified action or operation on the IRI device.

Parameters

- `iri_device` - IRI device data structure
- `action` - Actions or operations to start

This function is used to start inventory to read tags.

Return

`ipj_error`

`ipj_resume`

Note: This feature is not available in this release and will return a ...*`_NOT_IMPLEMENTED`* error code if called.

`ipj_error ipj_resume(ipj_iri_device * iri_device, ipj_action action)`

This function resumes the specified action or operation on the IRI device.

Parameters

- `iri_device` - IRI device data structure
- `action` - Actions or operations to be resumed

This function is used to resume operations after the IRI device has halted.

Return

`ipj_error`

ipj_stop

ipj_error **ipj_stop**(*ipj_iri_device* * iri_device, ipj_action action)

This function stops the specified action or operation on the IRI device.

Parameters

- *iri_device* - IRI device data structure
- *action* - Actions or operations to be stopped

This function is used to stop inventory to read tags.

Return

ipj_error

ipj_receive

ipj_error **ipj_receive**(*ipj_iri_device* * iri_device)

This function processes incoming reports from the IRI device.

Parameters

- *iri_device* - IRI device data structure

User calls this function to process incoming data. This function requests available data from the platform receive handler, buffers data, and checks for complete reports. This function returns immediately if there is not sufficient data for a report. Once a complete report is received, this function calls the report handler before returning.

User decides when to call this function. For example, User can poll or wait for an interrupt from the system. User responsibility to call this function to keep the receive buffer from overflowing.

Return

ipj_error

ipj_flash_handle_loader_block

ipj_error **ipj_flash_handle_loader_block**(*ipj_iri_device* * iri_device, uint32_t len, uint8_t * data)

This function takes in an IRI_Loader chunk and processes it accordingly.

Parameters

- *iri_device* - IRI device data structure
- *len* - Length of the data chunk (in bytes)
- *data* - Pointer to the data chunk to be processed

Return

ipj_error

IPJ_CLEAR_STRUCT

void **IPJ_CLEAR_STRUCT** (struct);

IPJ_CLEAR_STRUCT is a convenience macro provided to clear out IRI related structures before use.

We recommend using this macro on every data structure passed into an IRI function to ensure proper behavior.

1.5.9 Structures

Key Code Related Structures

ipj_key_info

struct **ipj_key_info**

#include <iri.h>

Public Members

bool **has_key_type**

ipj_key_type **key_type**

bool **has_count**

uint32_t **count**

bool **has_length**

uint32_t **length**

bool **has_key_permissions**

ipj_key_permissions **key_permissions**

ipj_key_list

struct **ipj_key_list**

#include <iri.h>

Public Members

bool **has_key**

ipj_key key

size_t list_count

uint32_t list[32]

bool has_bank_index

uint32_t bank_index

bool has_value_index

uint32_t value_index

bool has_length

uint32_t length

`ipj_key_value`

struct ipj_key_value

#include <iri.h>

Public Members

bool has_key

ipj_key key

bool has_value

uint32_t value

bool has_bank_index

uint32_t bank_index

bool **has_value_index**

uint32_t **value_index**

Report Related Structures

`ipj_error_report`

struct **ipj_error_report**

#include <iri.h>

Public Members

bool **has_error**

ipj_error **error**

bool **has_param1**

uint32_t **param1**

bool **has_param2**

uint32_t **param2**

bool **has_param3**

uint32_t **param3**

bool **has_param4**

uint32_t **param4**

bool **has_timestamp**

uint64_t **timestamp**

size_t **lt_buffer_count**

uint32_t **lt_buffer**[5]

Note: timestamp units = milliseconds

`ipj_tag_operation_report`

struct **ipj_tag_operation_report**

#include <iri.h>

Public Members

bool **has_error**

ipj_error **error**

bool **has_tag**

ipj_tag **tag**

bool **has_tag_operation_type**

ipj_tag_operation_type **tag_operation_type**

bool **has_tag_operation_data**

ipj_tag_operation_data_t **tag_operation_data**

bool **has_retries**

uint32_t **retries**

bool **has_diagnostic**

uint32_t **diagnostic**

bool **has_timestamp**

uint64_t **timestamp**

size_t **lt_buffer_count**

uint32_t **lt_buffer**[30]

Note: timestamp units = milliseconds

ipj_tag

struct **ipj_tag**
#include <iri.h>

Public Members

bool **has_epc**

ipj_epc_t **epc**

bool **has_tid**

ipj_tid_t **tid**

bool **has_pc**

uint32_t **pc**

bool **has_xpc**

uint32_t **xpc**

bool **has_crc**

uint32_t **crc**

bool **has_timestamp**

uint64_t **timestamp**

bool **has_rssi**

int32_t **rssi**

bool **has_phase**

int32_t **phase**

bool **has_channel**

uint32_t **channel**

bool **has_antenna**

uint32_t **antenna**

Note: timestamp units = milliseconds, rssi units = centi-dBm, phase units = $2 * \text{Pi} / 128$ radians

`ipj_epc_t`

```
struct ipj_epc_t
#include <iri.h>
```

Public Members

size_t **size**

uint8_t **bytes**[64]

`ipj_tid_t`

```
struct ipj_tid_t
#include <iri.h>
```

Public Members

size_t **size**

uint8_t **bytes**[48]

`ipj_tag_operation_data_t`

struct **ipj_tag_operation_data_t**

#include <iri.h>

Public Members

size_t **size**

uint8_t **bytes**[64]

`ipj_stop_report`

struct **ipj_stop_report**

#include <iri.h>

Public Members

bool **has_error**

ipj_error **error**

bool **has_action**

ipj_action **action**

bool **has_timestamp**

uint64_t **timestamp**

size_t **lt_buffer_count**

uint32_t **lt_buffer**[2]

Note: timestamp units = milliseconds

`ipj_status_report`

struct **ipj_status_report**

#include <iri.h>

Public Members

bool **has_status_flag**

ipj_status_flag **status_flag**

bool **has_timestamp**

uint64_t **timestamp**

bool **has_status_1**

uint32_t **status_1**

bool **has_status_2**

uint32_t **status_2**

bool **has_status_3**

uint32_t **status_3**

size_t **data_count**

uint32_t **data**[16]

size_t **lt_buffer_count**

uint32_t **lt_buffer**[20]

Note: timestamp units = milliseconds

IRI Connection Structures

`ipj_serial_connection`

struct **ipj_serial_connection**

#include <iri.h>

Public Members

`ipj_baud_rate` **baudrate**

`ipj_parity` **parity**

`ipj_connection_params`

Note: This union holds one of each type of connection parameter struct. In the future this will allow connection parameters for a variety of connection mediums to be set

IRI Device Structure

`ipj_iri_device`

struct **ipj_iri_device**

#include <iri.h>

User application allocates or declares structure before connecting or communicating with the IRI device.

Public Members

IPJ_READER_CONTEXT **reader_context**

PLATFORM_OPEN_PORT_HANDLER_EXT **platform_open_port_handler**

`void *` **platform_open_port_args**

PLATFORM_CLOSE_PORT_HANDLER_EXT **platform_close_port_handler**

`void *` **platform_close_port_args**

PLATFORM_TRANSMIT_HANDLER_EXT **platform_transmit_handler**

void * **platform_transmit_args**

PLATFORM_RECEIVE_HANDLER_EXT **platform_receive_handler**

void * **platform_receive_args**

PLATFORM_TIMESTAMP_MS_HANDLER_EXT **platform_timestamp_ms_handler**

void * **platform_timestamp_ms_args**

PLATFORM_SLEEP_MS_HANDLER_EXT **platform_sleep_ms_handler**

void * **platform_sleep_ms_args**

PLATFORM_FLUSH_PORT_HANDLER_EXT **platform_flush_port_handler**

void * **platform_flush_port_args**

PLATFORM_MODIFY_CONNECTION_HANDLER_EXT **plat-**
form_modify_connection_handler

void * **platform_modify_connection_args**

IPJ_READER_IDENTIFIER **reader_identifier**

IPJ_READER_USER_IDENTIFIER **reader_user_identifier**

REPORT_HANDLER_EXT **report_handler**

void * **report_args**

DIAGNOSTIC_HANDLER_EXT **diagnostic_handler**

void * **diagnostic_args**

uint32_t **transmit_timeout_ms**

uint32_t **receive_timeout_ms**

bool **initialized**

uint8_t **protocol_flags**

uint8_t **rx_frame_sync_count**

uint8_t **tx_frame_sync_count**

uint8_t **sync_state**

uint32_t **frame_length**

uint8_t **transmit_buffer**[IPJ_TRANSMIT_BUFFER_SIZE]

uint32_t **receive_index**

uint8_t **receive_buffer**[IPJ_RECEIVE_BUFFER_SIZE]

1.5.10 Defines

`ipj_action`

Define	Decimal	Hex
E_IPJ_ACTION_ALL	0	0x0
E_IPJ_ACTION_INVENTORY	1	0x1
E_IPJ_ACTION_TEST	2	0x2
E_IPJ_ACTION_GPIO	3	0x3
E_IPJ_ACTION_CLEAR_ERROR	4	0x4
E_IPJ_ACTION_STANDBY	6	0x6
E_IPJ_ACTION_SLEEP	7	0x7
E_IPJ_ACTION_NONE	15	0xF

ipj_baud_rate

Define	Decimal	Hex
E_IPJ_BAUD_RATE_BR110	110	0x6E
E_IPJ_BAUD_RATE_BR300	300	0x12C
E_IPJ_BAUD_RATE_BR600	600	0x258
E_IPJ_BAUD_RATE_BR1200	1200	0x4B0
E_IPJ_BAUD_RATE_BR2400	2400	0x960
E_IPJ_BAUD_RATE_BR4800	4800	0x12C0
E_IPJ_BAUD_RATE_BR9600	9600	0x2580
E_IPJ_BAUD_RATE_BR14400	14400	0x3840
E_IPJ_BAUD_RATE_BR19200	19200	0x4B00
E_IPJ_BAUD_RATE_BR38400	38400	0x9600
E_IPJ_BAUD_RATE_BR57600	57600	0xE100
E_IPJ_BAUD_RATE_BR115200	115200	0x1C200
E_IPJ_BAUD_RATE_BR230400	230400	0x38400
E_IPJ_BAUD_RATE_BR460800	460800	0x70800
E_IPJ_BAUD_RATE_BR921600	921600	0xE1000

ipj_blockpermalock_action

Define	Decimal	Hex
E_IPJ_BLOCKPERMALOCK_ACTION_READ	0	0x0
E_IPJ_BLOCKPERMALOCK_ACTION_PERMALOCK	1	0x1

Key Reference:

E_IPJ_KEY_BLOCKPERMALOCK_ACTION

ipj_connection_type

Define	Decimal	Hex
E_IPJ_CONNECTION_TYPE_SERIAL	0	0x0

ipj_error

Define	Decimal	Hex
E_IPJ_ERROR_SUCCESS	0	0x0
E_IPJ_ERROR_GENERAL_ERROR	1	0x1
E_IPJ_ERROR_SET_KEY_INVALID	2	0x2
E_IPJ_ERROR_SET_KEY_READ_ONLY	3	0x3
E_IPJ_ERROR_SET_KEY_OUT_OF_RANGE	4	0x4
E_IPJ_ERROR_GET_KEY_INVALID	5	0x5
E_IPJ_ERROR_GET_KEY_WRITE_ONLY	6	0x6
E_IPJ_ERROR_COMMAND_INVALID	7	0x7
E_IPJ_ERROR_COMMAND_START_FAILURE	8	0x8
E_IPJ_ERROR_COMMAND_DECODE_FAILURE	9	0x9
E_IPJ_ERROR_COMMAND_ENCODE_FAILURE	10	0xA
E_IPJ_ERROR_COMMAND_STALLED	11	0xB

Continued on next page

Table 1.8 – continued from previous page

Define	Decimal	Hex
E_IPJ_ERROR_VALUE_INVALID	12	0xC
E_IPJ_ERROR_MORE_THAN_ONE_COMMAND_RECEIVED	13	0xD
E_IPJ_ERROR_NOT_IMPLEMENTED	14	0xE
E_IPJ_ERROR_INVALID_PRODUCT_CONFIGURATION	15	0xF
E_IPJ_ERROR_INVALID_FACTORY_SETTINGS	16	0x10
E_IPJ_ERROR_RESPONSE_ENCODE_FAILURE	17	0x11
E_IPJ_ERROR_COMMAND_VERIFY_FAILURE	18	0x12
E_IPJ_ERROR_INTERNAL_NON_RECOVERABLE	19	0x13
E_IPJ_ERROR_TEMPLATE_DECODE_FAILURE	20	0x14
E_IPJ_ERROR_SYSTEM_IN_ERROR_STATE	21	0x15
E_IPJ_ERROR_TEST_ERROR	22	0x16
E_IPJ_ERROR_STORED_SETTING_DECODE	23	0x17
E_IPJ_ERROR_VALUE_INDEX_OUT_OF_RANGE	24	0x18
E_IPJ_ERROR_BANK_INDEX_OUT_OF_RANGE	25	0x19
E_IPJ_ERROR_INVALID_PRODUCT_CALIBRATION	26	0x1A
E_IPJ_ERROR_REPORT_SIZE_WOULD_OVERFLOW	27	0x1B
E_IPJ_ERROR_GEN2_TAG_OTHER_ERROR	16777217	0x1000001
E_IPJ_ERROR_GEN2_TAG_MEMORY_OVERRUN	16777218	0x1000002
E_IPJ_ERROR_GEN2_TAG_MEMORY_LOCKED	16777219	0x1000003
E_IPJ_ERROR_GEN2_TAG_INSUFFICIENT_POWER	16777220	0x1000004
E_IPJ_ERROR_GEN2_TAG_NON_SPECIFIC_ERROR	16777221	0x1000005
E_IPJ_ERROR_API_DEVICE_NOT_INITIALIZED	33554433	0x2000001
E_IPJ_ERROR_API_SERIAL_PORT_ERROR	33554434	0x2000002
E_IPJ_ERROR_API_CONNECTION_READ_TIMEOUT	33554435	0x2000003
E_IPJ_ERROR_API_CONNECTION_WRITE_TIMEOUT	33554436	0x2000004
E_IPJ_ERROR_API_CONNECTION_WRITE_ERROR	33554437	0x2000005
E_IPJ_ERROR_API_RX_BUFF_TOO_SMALL	33554438	0x2000006
E_IPJ_ERROR_API_MESSAGE_INVALID	33554439	0x2000007
E_IPJ_ERROR_API_NO_HANDLER	33554440	0x2000008
E_IPJ_ERROR_API_INVALID_LOADER_BLOCK	33554441	0x2000009
E_IPJ_ERROR_API_RESPONSE_MISMATCH	33554442	0x200000A
E_IPJ_ERROR_API_INVALID_PARAMETER	33554443	0x200000B
E_IPJ_ERROR_API_NON_LT_PACKET_DETECTED	33554444	0x200000C
E_IPJ_ERROR_IRI_FRAME_DROPPED	50331649	0x3000001
E_IPJ_ERROR_IRI_FRAME_INVALID	50331650	0x3000002
E_IPJ_ERROR_MAC_GENERAL	67108865	0x4000001
E_IPJ_ERROR_MAC_CRC_MISMATCH	67108866	0x4000002
E_IPJ_ERROR_MAC_NO_TAG_RESPONSE	67108867	0x4000003
E_IPJ_ERROR_MAC_TAG_LOST	67108868	0x4000004
E_IPJ_ERROR_BTS_DEVICE_WATCHDOG_RESET	83886081	0x5000001
E_IPJ_ERROR_BTS_VALUE_INVALID	83886082	0x5000002
E_IPJ_ERROR_BTS_FLASH_WRITE	83886083	0x5000003
E_IPJ_ERROR_BTS_FLASH_READ	83886084	0x5000004
E_IPJ_ERROR_BTS_FLASH_ADDRESS	83886085	0x5000005
E_IPJ_ERROR_BTS_FLASH_ERASE	83886086	0x5000006
E_IPJ_ERROR_BTS_UNKNOWN_COMMAND	83886087	0x5000007
E_IPJ_ERROR_BTS_COMMAND_DECODE_FAILURE	83886088	0x5000008
E_IPJ_ERROR_TRANSCEIVER_FAILURE	100663297	0x6000001
E_IPJ_ERROR_LIMIT_PA_TEMPERATURE_MAX	117440513	0x7000001

ipj_handler_type

Define	Decimal	Hex
E_IPJ_HANDLER_TYPE_PLATFORM_OPEN_PORT	0	0x0
E_IPJ_HANDLER_TYPE_PLATFORM_CLOSE_PORT	1	0x1
E_IPJ_HANDLER_TYPE_PLATFORM_TRANSMIT	2	0x2
E_IPJ_HANDLER_TYPE_PLATFORM_RECEIVE	3	0x3
E_IPJ_HANDLER_TYPE_PLATFORM_TIMESTAMP	4	0x4
E_IPJ_HANDLER_TYPE_PLATFORM_SLEEP_MS	5	0x5
E_IPJ_HANDLER_TYPE_REPORT	6	0x6
E_IPJ_HANDLER_TYPE_DIAGNOSTIC	7	0x7
E_IPJ_HANDLER_TYPE_PLATFORM_MODIFY_CONNECTION	8	0x8
E_IPJ_HANDLER_TYPE_PLATFORM_FLUSH_PORT	9	0x9

ipj_inventory_search_mode

Define	Decimal	Hex
E_IPJ_INVENTORY_SEARCH_MODE_AUTO_SEARCH	0	0x0
E_IPJ_INVENTORY_SEARCH_MODE_DUAL_TARGET	1	0x1
E_IPJ_INVENTORY_SEARCH_MODE_SINGLE_TARGET_A_TO_B	2	0x2
E_IPJ_INVENTORY_SEARCH_MODE_SINGLE_TARGET_B_TO_A	3	0x3

Key Reference:

*E_IPJ_KEY_INVENTORY_SEARCH_MODE***ipj_inventory_select_flag**

Define	Decimal	Hex
E_IPJ_INVENTORY_SELECT_FLAG_AUTO_SL	0	0x0
E_IPJ_INVENTORY_SELECT_FLAG_ALL_SL	1	0x1
E_IPJ_INVENTORY_SELECT_FLAG_NOT_SL	2	0x2
E_IPJ_INVENTORY_SELECT_FLAG_SL	3	0x3

Key Reference:

*E_IPJ_KEY_INVENTORY_SELECT_FLAG***ipj_key**

Define	Decimal	Hex
E_IPJ_KEY_BOOTSTRAP_VERSION	1	0x1
E_IPJ_KEY_BOOTSTRAP_CRC	2	0x2
E_IPJ_KEY_APPLICATION_VERSION	3	0x3
E_IPJ_KEY_APPLICATION_CRC	4	0x4
E_IPJ_KEY_SECONDARY_IMAGE_VERSION	5	0x5
E_IPJ_KEY_SECONDARY_IMAGE_CRC	6	0x6
E_IPJ_KEY_SECONDARY_IMAGE_TYPE	7	0x7
E_IPJ_KEY_APPLICATION_REVISION_ID	8	0x8
E_IPJ_KEY_APPLICATION_BUILD_ID	9	0x9
E_IPJ_KEY_PRODUCT_ID	10	0xA

Continued on next page

Table 1.9 – continued from previous page

Define	Decimal	Hex
E_IPJ_KEY_SERIAL_NUMBER	11	0xB
E_IPJ_KEY_TRANSCEIVER_ID	12	0xC
E_IPJ_KEY_MICROPROCESSOR_ID	13	0xD
E_IPJ_KEY_CUSTOMER_VERSION	14	0xE
E_IPJ_KEY_CUSTOMER_ID	15	0xF
E_IPJ_KEY_CUSTOMER_PRODUCT_ID	16	0x10
E_IPJ_KEY_CALIBRATION_INFO	17	0x11
E_IPJ_KEY_TEST_INFO	18	0x12
E_IPJ_KEY_PRODUCT_SKU	19	0x13
E_IPJ_KEY_LOT_DATE_CODE	20	0x14
E_IPJ_KEY_PRODUCT_KEY	21	0x15
E_IPJ_KEY_SECONDARY_IMAGE_LOCATION	22	0x16
E_IPJ_KEY_SECONDARY_IMAGE_SIZE	23	0x17
E_IPJ_KEY_UNIQUE_ID	24	0x18
E_IPJ_KEY_HARDWARE_REVISION	25	0x19
E_IPJ_KEY_REGION_ID	32	0x20
E_IPJ_KEY_REGION_CHANNEL_TABLE	33	0x21
E_IPJ_KEY_REGION_CHANNEL_TABLE_SIZE	34	0x22
E_IPJ_KEY_REGION_ON_TIME_NOMINAL	35	0x23
E_IPJ_KEY_REGION_ON_TIME_ACCESS	36	0x24
E_IPJ_KEY_REGION_OFF_TIME	37	0x25
E_IPJ_KEY_REGION_OFF_TIME_SAME_CHANNEL	38	0x26
E_IPJ_KEY_REGION_START_FREQUENCY_KHZ	39	0x27
E_IPJ_KEY_REGION_CHANNEL_SPACING_KHZ	40	0x28
E_IPJ_KEY_REGION_RANDOM_HOP	41	0x29
E_IPJ_KEY_REGION_INDUCE_PLL_R_DIVIDER	42	0x2A
E_IPJ_KEY_REGION_RF_FILTER	43	0x2B
E_IPJ_KEY_ANTENNA_TX_POWER	49	0x31
E_IPJ_KEY_ANTENNA_SEQUENCE	50	0x32
E_IPJ_KEY_INVENTORY_TAG_POPULATION	64	0x40
E_IPJ_KEY_INVENTORY_SELECT_FLAG	65	0x41
E_IPJ_KEY_INVENTORY_SESSION	66	0x42
E_IPJ_KEY_INVENTORY_SEARCH_MODE	67	0x43
E_IPJ_KEY_FAST_ID_ENABLE	69	0x45
E_IPJ_KEY_TAG_FOCUS_ENABLE	70	0x46
E_IPJ_KEY_TAG_OPERATION_ENABLE	71	0x47
E_IPJ_KEY_TAG_OPERATION_RETRIES	72	0x48
E_IPJ_KEY_SELECT_ENABLE	80	0x50
E_IPJ_KEY_SELECT_TARGET	81	0x51
E_IPJ_KEY_SELECT_ACTION	82	0x52
E_IPJ_KEY_SELECT_MEM_BANK	83	0x53
E_IPJ_KEY_SELECT_POINTER	84	0x54
E_IPJ_KEY_SELECT_MASK_LENGTH	85	0x55
E_IPJ_KEY_SELECT_MASK_VALUE	86	0x56
E_IPJ_KEY_TAG_OPERATION	96	0x60
E_IPJ_KEY_ACCESS_PASSWORD	97	0x61
E_IPJ_KEY_KILL_PASSWORD	98	0x62
E_IPJ_KEY_READ_MEM_BANK	99	0x63
E_IPJ_KEY_READ_WORD_POINTER	100	0x64

Continued on next page

Table 1.9 – continued from previous page

Define	Decimal	Hex
E_IPJ_KEY_READ_WORD_COUNT	101	0x65
E_IPJ_KEY_WRITE_MEM_BANK	102	0x66
E_IPJ_KEY_WRITE_WORD_POINTER	103	0x67
E_IPJ_KEY_WRITE_WORD_COUNT	104	0x68
E_IPJ_KEY_WRITE_DATA	105	0x69
E_IPJ_KEY_LOCK_PAYLOAD	106	0x6A
E_IPJ_KEY_BLOCKPERMALOCK_ACTION	107	0x6B
E_IPJ_KEY_BLOCKPERMALOCK_MEM_BANK	108	0x6C
E_IPJ_KEY_BLOCKPERMALOCK_BLOCK_POINTER	109	0x6D
E_IPJ_KEY_BLOCKPERMALOCK_BLOCK_RANGE	110	0x6E
E_IPJ_KEY_BLOCKPERMALOCK_MASK	111	0x6F
E_IPJ_KEY_WRITE_EPC_LENGTH_CONTROL	112	0x70
E_IPJ_KEY_WRITE_EPC_LENGTH_VALUE	113	0x71
E_IPJ_KEY_WRITE_EPC_AFI_CONTROL	114	0x72
E_IPJ_KEY_WRITE_EPC_AFI_VALUE	115	0x73
E_IPJ_KEY_QT_ACTION	116	0x74
E_IPJ_KEY_QT_PERSISTENCE	117	0x75
E_IPJ_KEY_QT_DATA_PROFILE	118	0x76
E_IPJ_KEY_QT_ACCESS_RANGE	119	0x77
E_IPJ_KEY_QT_TAG_OPERATION	120	0x78
E_IPJ_KEY_AUTOSTOP_DURATION_MS	137	0x89
E_IPJ_KEY_AUTOSTOP_TAG_COUNT	139	0x8B
E_IPJ_KEY_REPORT_CONTROL_TAG	161	0xA1
E_IPJ_KEY_REPORT_CONTROL_STATUS	162	0xA2
E_IPJ_KEY_REPORT_CONTROL_TIMESTAMP	163	0xA3
E_IPJ_KEY_RESPONSE_CONTROL_TIMESTAMP	164	0xA4
E_IPJ_KEY_GPIO_MODE	192	0xC0
E_IPJ_KEY_GPIO_STATE	193	0xC1
E_IPJ_KEY_GPIO_HI_ACTION	194	0xC2
E_IPJ_KEY_GPIO_LO_ACTION	195	0xC3
E_IPJ_KEY_GPIO_DEBOUNCE_MS	197	0xC5
E_IPJ_KEY_GPIO_CURRENT_STATE	198	0xC6
E_IPJ_KEY_RF_MODE	208	0xD0
E_IPJ_KEY_FIRST_ERROR	224	0xE0
E_IPJ_KEY_LAST_ERROR	225	0xE1
E_IPJ_KEY_SYSTEM_ERROR	226	0xE2
E_IPJ_KEY_DEVICE_BAUDRATE	256	0x100
E_IPJ_KEY_DEVICE_IDLE_POWER_MODE	257	0x101
E_IPJ_KEY_ONBOOT_START_ACTION	258	0x102
E_IPJ_KEY_ENABLE_LT_REPORTS	259	0x103
E_IPJ_KEY_TEST_ID	1024	0x400
E_IPJ_KEY_TEST_PARAMETERS	1025	0x401
E_IPJ_KEY_TEST_RESULT_1	1026	0x402
E_IPJ_KEY_TEST_RESULT_2	1027	0x403
E_IPJ_KEY_TEST_RESULT_3	1028	0x404
E_IPJ_KEY_TEST_DATA	1029	0x405
E_IPJ_KEY_TEST_FREQUENCY	1030	0x406
E_IPJ_KEY_TEST_POWER	1031	0x407
E_IPJ_KEY_TEST_RF_MODE	1032	0x408

Continued on next page

Table 1.9 – continued from previous page

Define	Decimal	Hex
E_IPJ_KEY_TEST_TIME	1033	0x409
E_IPJ_KEY_TEST_EVENT	1034	0x40A
E_IPJ_KEY_TEST_REPORTS	1035	0x40B
E_IPJ_KEY_TEST_SYSTEM	1036	0x40C
E_IPJ_KEY_TEST_DEBUG_PORT	1037	0x40D
E_IPJ_KEY_GENERIC_DATA	3072	0xC00
E_IPJ_KEY_OEM_DATA	3073	0xC01

ipj_key_permissions

Define	Decimal	Hex
E_IPJ_KEY_PERMISSIONS_READ_ONLY	0	0x0
E_IPJ_KEY_PERMISSIONS_WRITE_ONLY	1	0x1
E_IPJ_KEY_PERMISSIONS_READ_WRITE	2	0x2

ipj_key_type

Define	Decimal	Hex
E_IPJ_KEY_TYPE_UINT32	0	0x0
E_IPJ_KEY_TYPE_UINT16	1	0x1
E_IPJ_KEY_TYPE_UINT8	2	0x2
E_IPJ_KEY_TYPE_INT32	3	0x3
E_IPJ_KEY_TYPE_INT16	4	0x4
E_IPJ_KEY_TYPE_INT8	5	0x5
E_IPJ_KEY_TYPE_BOOL	6	0x6

ipj_mem_bank

Define	Decimal	Hex
E_IPJ_MEM_BANK_RESERVED	0	0x0
E_IPJ_MEM_BANK_EPC	1	0x1
E_IPJ_MEM_BANK_TID	2	0x2
E_IPJ_MEM_BANK_USER	3	0x3

Key Reference:

*E_IPJ_KEY_READ_MEM_BANK**E_IPJ_KEY_WRITE_MEM_BANK**E_IPJ_KEY_BLOCKPERMALOCK_MEM_BANK***ipj_parity**

Define	Decimal	Hex
E_IPJ_PARITY_PNONE	0	0x0
E_IPJ_PARITY_PEVEN	1	0x1
E_IPJ_PARITY_PODD	2	0x2

ipj_qt_access_range

Define	Decimal	Hex
E_IPJ_QT_ACCESS_RANGE_NORMAL	0	0x0
E_IPJ_QT_ACCESS_RANGE_SHORT	1	0x1

Key Reference:

E_IPJ_KEY_QT_ACCESS_RANGE

ipj_qt_action

Define	Decimal	Hex
E_IPJ_QT_ACTION_READ	0	0x0
E_IPJ_QT_ACTION_WRITE	1	0x1

Key Reference:

E_IPJ_KEY_QT_ACTION

ipj_qt_data_profile

Define	Decimal	Hex
E_IPJ_QT_DATA_PROFILE_PRIVATE	0	0x0
E_IPJ_QT_DATA_PROFILE_PUBLIC	1	0x1

Key Reference:

E_IPJ_KEY_QT_DATA_PROFILE

ipj_qt_persistence

Define	Decimal	Hex
E_IPJ_QT_PERSISTENCE_TEMPORARY	0	0x0
E_IPJ_QT_PERSISTENCE_PERMANENT	1	0x1

Key Reference:

E_IPJ_KEY_QT_PERSISTENCE

`ipj_region`

Define	Decimal	Hex
E_IPJ_REGION_FCC_PART_15_247	0	0x0
E_IPJ_REGION_HONG_KONG_920_925_MHZ	3	0x3
E_IPJ_REGION_TAIWAN_922_928_MHZ	4	0x4
E_IPJ_REGION_ETSI_EN_302_208_V1_4_1	7	0x7
E_IPJ_REGION_KOREA_917_921_MHZ	8	0x8
E_IPJ_REGION_MALAYSIA_919_923_MHZ	9	0x9
E_IPJ_REGION_CHINA_920_925_MHZ	10	0xA
E_IPJ_REGION_SOUTH_AFRICA_915_919_MHZ	12	0xC
E_IPJ_REGION_BRAZIL_902_907_AND_915_928_MHZ	13	0xD
E_IPJ_REGION_THAILAND_920_925_MHZ	14	0xE
E_IPJ_REGION_SINGAPORE_920_925_MHZ	15	0xF
E_IPJ_REGION_AUSTRALIA_920_926_MHZ	16	0x10
E_IPJ_REGION_INDIA_865_867_MHZ	17	0x11
E_IPJ_REGION_URUGUAY_916_928_MHZ	18	0x12
E_IPJ_REGION_VIETNAM_920_925_MHZ	19	0x13
E_IPJ_REGION_ISRAEL_915_917_MHZ	20	0x14
E_IPJ_REGION_PHILIPPINES_918_920_MHZ	21	0x15
E_IPJ_REGION_INDONESIA_923_925_MHZ	23	0x17
E_IPJ_REGION_NEW_ZEALAND_921P5_928_MHZ	24	0x18
E_IPJ_REGION_JAPAN_916_921_MHZ_NO_LBT	25	0x19
E_IPJ_REGION_PERU_916_928_MHZ	26	0x1A
E_IPJ_REGION_RUSSIA_916_921_MHZ	27	0x1B
E_IPJ_REGION_CUSTOM	256	0x100

Key Reference:

E_IPJ_KEY_REGION_ID

`ipj_report_id`

Define	Decimal	Hex
E_IPJ_REPORT_ID_TAG_OPERATION_REPORT	2	0x2
E_IPJ_REPORT_ID_STOP_REPORT	5	0x5
E_IPJ_REPORT_ID_TEST_REPORT	12	0xC
E_IPJ_REPORT_ID_ERROR_REPORT	13	0xD
E_IPJ_REPORT_ID_STATUS_REPORT	14	0xE
E_IPJ_REPORT_ID_GPIO_REPORT	15	0xF
E_IPJ_REPORT_ID_IDD_REPORT	16	0x10

`ipj_reset_type`

Define	Decimal	Hex
E_IPJ_RESET_TYPE_SOFT	0	0x0
E_IPJ_RESET_TYPE_TO_BOOTLOADER	1	0x1
E_IPJ_RESET_TYPE_FACTORY_RESTORE	2	0x2

ipj_select_action

Define	Decimal	Hex
E_IPJ_SELECT_ACTION_ASLINVA_DSLINVB	0	0x0
E_IPJ_SELECT_ACTION_ASLINVA_NOTHING	1	0x1
E_IPJ_SELECT_ACTION_NOTHING_DSLINVB	2	0x2
E_IPJ_SELECT_ACTION_NSLINVS_NOTHING	3	0x3
E_IPJ_SELECT_ACTION_DSLINVB_ASLINVA	4	0x4
E_IPJ_SELECT_ACTION_DSLINVB_NOTHING	5	0x5
E_IPJ_SELECT_ACTION_NOTHING_ASLINVA	6	0x6
E_IPJ_SELECT_ACTION_NOTHING_NSLINVS	7	0x7

Key Reference:

E_IPJ_KEY_SELECT_ACTION

ipj_select_target

Define	Decimal	Hex
E_IPJ_SELECT_TARGET_SESSION_S0	0	0x0
E_IPJ_SELECT_TARGET_SESSION_S1	1	0x1
E_IPJ_SELECT_TARGET_SESSION_S2	2	0x2
E_IPJ_SELECT_TARGET_SESSION_S3	3	0x3
E_IPJ_SELECT_TARGET_SL_FLAG	4	0x4
E_IPJ_SELECT_TARGET_RFU_1	5	0x5
E_IPJ_SELECT_TARGET_RFU_2	6	0x6
E_IPJ_SELECT_TARGET_RFU_3	7	0x7

Key Reference:

E_IPJ_KEY_SELECT_TARGET

ipj_status_flag

Define	Decimal	Hex
E_IPJ_STATUS_FLAG_BIT_CHANNEL_ACTIVITY	1	0x1
E_IPJ_STATUS_FLAG_BIT_IDD	2	0x2

Key Reference:

E_IPJ_KEY_REPORT_CONTROL_STATUS

ipj_tag_flag

Define	Decimal	Hex
E_IPJ_TAG_FLAG_BIT_EPC	1	0x1
E_IPJ_TAG_FLAG_BIT_TID	2	0x2
E_IPJ_TAG_FLAG_BIT_PC	4	0x4
E_IPJ_TAG_FLAG_BIT_XPC	8	0x8
E_IPJ_TAG_FLAG_BIT_CRC	16	0x10
E_IPJ_TAG_FLAG_BIT_TIMESTAMP	32	0x20
E_IPJ_TAG_FLAG_BIT_RSSI	64	0x40
E_IPJ_TAG_FLAG_BIT_PHASE	128	0x80
E_IPJ_TAG_FLAG_BIT_CHANNEL	256	0x100
E_IPJ_TAG_FLAG_BIT_ANTENNA	512	0x200
E_IPJ_TAG_FLAG_BIT_RETRIES	1024	0x400

Key Reference:

E_IPJ_KEY_REPORT_CONTROL_TAG

ipj_tag_operation_type

Define	Decimal	Hex
E_IPJ_TAG_OPERATION_TYPE_NONE	0	0x0
E_IPJ_TAG_OPERATION_TYPE_READ	1	0x1
E_IPJ_TAG_OPERATION_TYPE_WRITE	2	0x2
E_IPJ_TAG_OPERATION_TYPE_LOCK	3	0x3
E_IPJ_TAG_OPERATION_TYPE_KILL	4	0x4
E_IPJ_TAG_OPERATION_TYPE_BLOCKPERMALOCK	5	0x5
E_IPJ_TAG_OPERATION_TYPE_WRITE_EPC	6	0x6
E_IPJ_TAG_OPERATION_TYPE_QT	7	0x7
E_IPJ_TAG_OPERATION_TYPE_CUSTOM	8	0x8
E_IPJ_TAG_OPERATION_TYPE_CUSTOM_2	9	0x9

Key Reference:

E_IPJ_KEY_TAG_OPERATION

E_IPJ_KEY_QT_TAG_OPERATION

ipj_test_id

Define	Decimal	Hex
E_IPJ_TEST_ID_NONE	0	0x0
E_IPJ_TEST_ID_GENERIC	1	0x1
E_IPJ_TEST_ID_BIST	2	0x2
E_IPJ_TEST_ID_MEMORY_READ	3	0x3
E_IPJ_TEST_ID_MEMORY_WRITE	4	0x4
E_IPJ_TEST_ID_TRANSCEIVER_READ	5	0x5
E_IPJ_TEST_ID_TRANSCEIVER_WRITE	6	0x6
E_IPJ_TEST_ID_SET_FREQUENCY	7	0x7
E_IPJ_TEST_ID_CW_CONTROL	8	0x8
E_IPJ_TEST_ID_PRBS_CONTROL	9	0x9

Continued on next page

Table 1.10 – continued from previous page

Define	Decimal	Hex
E_IPJ_TEST_ID_GEN2_TX_CONTROL	10	0xA
E_IPJ_TEST_ID_TRANSCEIVER_READ_MODIFY_WRITE	11	0xB
E_IPJ_TEST_ID_CALIBRATE_BEGIN	101	0x65
E_IPJ_TEST_ID_CALIBRATE_END	102	0x66
E_IPJ_TEST_ID_CALIBRATE_FORWARD_POWER	103	0x67
E_IPJ_TEST_ID_CALIBRATE_REVERSE_POWER	104	0x68
E_IPJ_TEST_ID_CALIBRATE_DC_OFFSET	105	0x69
E_IPJ_TEST_ID_CALIBRATE_PA_BIAS	106	0x6A
E_IPJ_TEST_ID_CALIBRATE_GROSS_GAIN	107	0x6B
E_IPJ_TEST_ID_CALIBRATE_RSSI	108	0x6C
E_IPJ_TEST_ID_CALIBRATE_TEMPERATURE	109	0x6D
E_IPJ_TEST_ID_CALIBRATE_TX_PRE_DISTORTION	110	0x6E
E_IPJ_TEST_ID_CALIBRATE_SET_KEY	111	0x6F
E_IPJ_TEST_ID_CALIBRATE_FINALIZE_KEYS	112	0x70
E_IPJ_TEST_ID_HARDWARE_CONTROL	201	0xC9
E_IPJ_TEST_ID_INVENTORY_CONTROL	202	0xCA
E_IPJ_TEST_ID_GENERIC_TX_CONTROL	203	0xCB
E_IPJ_TEST_ID_POWER_SCALAR_CONTROL	204	0xCC
E_IPJ_TEST_ID_GROSS_GAIN_CONTROL	205	0xCD
E_IPJ_TEST_ID_PA_BIAS_CONTROL	206	0xCE
E_IPJ_TEST_ID_GENERIC_RX_CONTROL	207	0xCF
E_IPJ_TEST_ID_USER_DAC_CONTROL	208	0xD0
E_IPJ_TEST_ID_USER_ADC_CONTROL	209	0xD1
E_IPJ_TEST_ID_AUX_DAC_CONTROL	210	0xD2
E_IPJ_TEST_ID_AUX_ADC_CONTROL	211	0xD3
E_IPJ_TEST_ID_GPIO_CONTROL	212	0xD4
E_IPJ_TEST_ID_DEBUG_UART_CONTROL	213	0xD5
E_IPJ_TEST_ID_TEMPERATURE_CONTROL	214	0xD6
E_IPJ_TEST_ID_RF_PROFILE_CONTROL	215	0xD7
E_IPJ_TEST_ID_SENSOR_CONTROL	216	0xD8
E_IPJ_TEST_ID_CALIBRATION_CONTROL	217	0xD9
E_IPJ_TEST_ID_ERROR_CONTROL	218	0xDA
E_IPJ_TEST_ID_RF_SWITCH_CONTROL	219	0xDB
E_IPJ_TEST_ID_GPIO32_CONTROL	220	0xDC
E_IPJ_TEST_ID_DAC_CONTROL	221	0xDD
E_IPJ_TEST_ID_ADC_CONTROL	222	0xDE
E_IPJ_TEST_ID_DEBUG_INFO	223	0xDF

ipj_write_epc_length_control

Define	Decimal	Hex
E_IPJ_WRITE_EPC_LENGTH_CONTROL_AUTO	0	0x0
E_IPJ_WRITE_EPC_LENGTH_CONTROL_USER_VALUE	1	0x1
E_IPJ_WRITE_EPC_LENGTH_CONTROL_ZERO	2	0x2
E_IPJ_WRITE_EPC_LENGTH_CONTROL_NO_CHANGE	3	0x3

Key Reference:

[*E_IPJ_KEY_WRITE_EPC_LENGTH_CONTROL*](#)

ipj_product_id

Define	Decimal	Hex
E_IPJ_PRODUCT_ID_NONE	0	0x0
E_IPJ_PRODUCT_ID_RS500	1	0x1
E_IPJ_PRODUCT_ID_RESERVED_2	2	0x2
E_IPJ_PRODUCT_ID_RS2000	3	0x3

Key Reference:

E_IPJ_KEY_PRODUCT_ID

ipj_product_sku

Define	Decimal	Hex
E_IPJ_PRODUCT_SKU_NONE	0	0x0
E_IPJ_PRODUCT_SKU_1	1	0x1
E_IPJ_PRODUCT_SKU_2	2	0x2
E_IPJ_PRODUCT_SKU_3	3	0x3

Key Reference:

E_IPJ_KEY_PRODUCT_SKU

ipj_response_timestamp_flag

Define	Decimal	Hex
E_IPJ_RES_TIMESTAMP_FLAG_BIT_RESET	1	0x1
E_IPJ_RES_TIMESTAMP_FLAG_BIT_MODIFY_CONN	2	0x2
E_IPJ_RES_TIMESTAMP_FLAG_BIT_GET_INFO	4	0x4
E_IPJ_RES_TIMESTAMP_FLAG_BIT_BULK_SET	8	0x8
E_IPJ_RES_TIMESTAMP_FLAG_BIT_BULK_GET	16	0x10
E_IPJ_RES_TIMESTAMP_FLAG_BIT_START	32	0x20
E_IPJ_RES_TIMESTAMP_FLAG_BIT_RESUME	64	0x40
E_IPJ_RES_TIMESTAMP_FLAG_BIT_STOP	128	0x80
E_IPJ_RES_TIMESTAMP_FLAG_BIT_INVALID	256	0x100
E_IPJ_RES_TIMESTAMP_FLAG_BIT_FLASH	512	0x200

Key Reference:

E_IPJ_KEY_RESPONSE_CONTROL_TIMESTAMP

ipj_report_timestamp_flag

Define	Decimal	Hex
E_IPJ_REP_TIMESTAMP_FLAG_BIT_TAG_OP	1	0x1
E_IPJ_REP_TIMESTAMP_FLAG_BIT_STOP	2	0x2
E_IPJ_REP_TIMESTAMP_FLAG_BIT_TEST	4	0x4
E_IPJ_REP_TIMESTAMP_FLAG_BIT_ERROR	8	0x8
E_IPJ_REP_TIMESTAMP_FLAG_BIT_STATUS	16	0x10
E_IPJ_REP_TIMESTAMP_FLAG_BIT_GPIO	32	0x20

Key Reference:

E_IPJ_KEY_REPORT_CONTROL_TIMESTAMP

ipj_gpio_mode

Define	Decimal	Hex
E_IPJ_GPIO_MODE_DISABLED	0	0x0
E_IPJ_GPIO_MODE_INPUT	1	0x1
E_IPJ_GPIO_MODE_OUTPUT	2	0x2
E_IPJ_GPIO_MODE_OUTPUT_PULSE	3	0x3
E_IPJ_GPIO_MODE_INPUT_ACTION	4	0x4
E_IPJ_GPIO_MODE_OUTPUT_ACTION	5	0x5
E_IPJ_GPIO_MODE_OUTPUT_PULSE_ACTION	6	0x6

Description:

GPIO pulse mode not currently supported

Key Reference:

E_IPJ_KEY_GPIO_MODE

ipj_gpio_state

Define	Decimal	Hex
E_IPJ_GPIO_STATE_LO	0	0x0
E_IPJ_GPIO_STATE_HI	1	0x1
E_IPJ_GPIO_STATE_FLOAT	2	0x2

Key Reference:

E_IPJ_KEY_GPIO_STATE

ipj_gpi_action

Define	Decimal	Hex
E_IPJ_GPI_ACTION_NONE	0	0x0
E_IPJ_GPI_ACTION_START_INVENTORY	1	0x1
E_IPJ_GPI_ACTION_STOP_INVENTORY	2	0x2

ipj_idle_power_mode

Define	Decimal	Hex
E_IPJ_IDLE_POWER_MODE_STANDARD	0	0x0
E_IPJ_IDLE_POWER_MODE_LOW_LATENCY	1	0x1

Key Reference:

E_IPJ_KEY_DEVICE_IDLE_POWER_MODE

1.5.11 Key Codes

This section describes parameters configured using *ipj_set*, *ipj_set_value*, or *ipj_bulk_set* IRI API functions, or retrieved using *ipj_get*, *ipj_get_value*, and *ipj_bulk_get* IRI API functions.

Key Codes By Category

Antenna

Key	Key Id	Range/Type	Banks/Values	Units	R/W
<i>E_IPJ_KEY_ANTENNA_TX_POWER</i>	49	[0,3150]	1/1	cdBm	R/W
<i>E_IPJ_KEY_ANTENNA_SEQUENCE</i>	50	uint8	0/16	–	R/W

Boot Action

Key	Key Id	Range/Type	Banks/Values	Units	R/W
<i>E_IPJ_KEY_ONBOOT_START_ACTION</i>	258	<i>ipj_action</i>	0/1	–	R/W

Device

Key	Key Id	Range/Type	Banks/Values	Units	R/W
<i>E_IPJ_KEY_DEVICE_BAUDRATE</i>	256	<i>ipj_baud_rate</i>	0/1	–	R/W

Error

Key	Key Id	Range/Type	Banks/Values	Units	R/W
<i>E_IPJ_KEY_FIRST_ERROR</i>	224	<i>ipj_error</i>	0/5	–	R
<i>E_IPJ_KEY_LAST_ERROR</i>	225	<i>ipj_error</i>	0/5	–	R
<i>E_IPJ_KEY_SYSTEM_ERROR</i>	226	<i>ipj_error</i>	0/5	–	R

GPIO

Key	Key Id	Range/Type	Banks/Values	Units	R/W
<i>E_IPJ_KEY_GPIO_MODE</i>	192	<i>ipj_gpio_mode</i>	5/1	–	R/W
<i>E_IPJ_KEY_GPIO_STATE</i>	193	<i>ipj_gpio_state</i>	5/1	–	R/W
<i>E_IPJ_KEY_GPIO_HI_ACTION</i>	194	<i>ipj_gpi_action</i>	5/1	–	R/W
<i>E_IPJ_KEY_GPIO_LO_ACTION</i>	195	<i>ipj_gpi_action</i>	5/1	–	R/W
<i>E_IPJ_KEY_GPIO_DEBOUNCE_MS</i>	197	uint32	5/1	ms	R/W
<i>E_IPJ_KEY_GPIO_CURRENT_STATE</i>	198	<i>ipj_gpio_state</i>	5/1	–	R

Generic

Key	Key Id	Range/Type	Banks/Values	Units	R/W
<i>E_IPJ_KEY_GENERIC_DATA</i>	3072	uint32	1/16	–	R/W
<i>E_IPJ_KEY_OEM_DATA</i>	3073	uint32	0/16	–	R

Info Only

Key	Key Id	Range/Type	Banks/Values	Units	R/W
<i>E_IPJ_KEY_BOOTSTRAP_VERSION</i>	1	uint32	0/1	–	R
<i>E_IPJ_KEY_BOOTSTRAP_CRC</i>	2	uint32	0/1	–	R
<i>E_IPJ_KEY_APPLICATION_VERSION</i>	3	uint32	0/1	–	R
<i>E_IPJ_KEY_APPLICATION_CRC</i>	4	uint32	0/1	–	R
<i>E_IPJ_KEY_SECONDARY_IMAGE_VERSION</i>	5	uint32	3/1	–	R
<i>E_IPJ_KEY_SECONDARY_IMAGE_CRC</i>	6	uint32	3/1	–	R
<i>E_IPJ_KEY_SECONDARY_IMAGE_TYPE</i>	7	uint32	3/1	–	R
<i>E_IPJ_KEY_APPLICATION_REVISION_ID</i>	8	uint32	0/1	–	R
<i>E_IPJ_KEY_APPLICATION_BUILD_ID</i>	9	uint32	0/1	–	R
<i>E_IPJ_KEY_PRODUCT_ID</i>	10	<i>ipj_product_id</i>	0/1	–	R
<i>E_IPJ_KEY_SERIAL_NUMBER</i>	11	uint32	0/1	–	R
<i>E_IPJ_KEY_TRANSCEIVER_ID</i>	12	uint32	0/1	–	R
<i>E_IPJ_KEY_MICROPROCESSOR_ID</i>	13	uint32	0/4	–	R
<i>E_IPJ_KEY_CUSTOMER_VERSION</i>	14	uint32	0/1	–	R
<i>E_IPJ_KEY_CUSTOMER_ID</i>	15	uint32	0/1	–	R
<i>E_IPJ_KEY_CUSTOMER_PRODUCT_ID</i>	16	uint32	0/1	–	R
<i>E_IPJ_KEY_CALIBRATION_INFO</i>	17	uint32	0/1	–	R
<i>E_IPJ_KEY_TEST_INFO</i>	18	uint32	0/1	–	R
<i>E_IPJ_KEY_PRODUCT_SKU</i>	19	<i>ipj_product_sku</i>	0/1	–	R
<i>E_IPJ_KEY_LOT_DATE_CODE</i>	20	uint32	0/1	–	R
<i>E_IPJ_KEY_PRODUCT_KEY</i>	21	uint32	0/1	–	R
<i>E_IPJ_KEY_SECONDARY_IMAGE_LOCATION</i>	22	uint32	3/1	–	R
<i>E_IPJ_KEY_SECONDARY_IMAGE_SIZE</i>	23	uint32	3/1	–	R
<i>E_IPJ_KEY_UNIQUE_ID</i>	24	uint32	0/2	–	R
<i>E_IPJ_KEY_HARDWARE_REVISION</i>	25	uint8	0/1	–	R

Inventory

Key	Key Id	Range/Type	Banks/Values	Units	R/W
<i>E_IPJ_KEY_INVENTORY_TAG_POPULATION</i>	64	uint32	0/1	–	R/W
<i>E_IPJ_KEY_INVENTORY_SELECT_FLAG</i>	65	<i>ipj_inventory_select_flag</i>	0/1	–	R/W
<i>E_IPJ_KEY_INVENTORY_SESSION</i>	66	[0,3]	0/1	–	R/W
<i>E_IPJ_KEY_INVENTORY_SEARCH_MODE</i>	67	<i>ipj_inventory_search_mode</i>	0/1	–	R/W
<i>E_IPJ_KEY_FAST_ID_ENABLE</i>	69	bool	0/1	–	R/W
<i>E_IPJ_KEY_TAG_FOCUS_ENABLE</i>	70	bool	0/1	–	R/W
<i>E_IPJ_KEY_TAG_OPERATION_ENABLE</i>	71	bool	0/1	–	R/W
<i>E_IPJ_KEY_TAG_OPERATION_RETRIES</i>	72	uint8	0/1	–	R/W
<i>E_IPJ_KEY_AUTOSTOP_DURATION_MS</i>	137	uint32	0/1	ms	R/W
<i>E_IPJ_KEY_AUTOSTOP_TAG_COUNT</i>	139	uint32	0/1	–	R/W

Power Mode

Key	Key Id	Range/Type	Banks/Values	Units	R/W
<i>E_IPJ_KEY_DEVICE_IDLE_POWER_MODE</i>	257	<i>ipj_idle_power_mode</i>	0/1	–	R/W

RF Mode

Key	Key Id	Range/Type	Banks/Values	Units	R/W
<i>E_IPJ_KEY_RF_MODE</i>	208	[0,4]	0/1	–	R/W

Region

Key	Key Id	Range/Type	Banks/Values	Units	R/W
<i>E_IPJ_KEY_REGION_ID</i>	32	<i>ipj_region</i>	0/1	–	R/W

Region - Custom

Key	Key Id	Range/Type	Banks/Values	Units	R/W
<i>E_IPJ_KEY_REGION_CHANNEL_TABLE</i>	33	uint8	0/50	–	R/W
<i>E_IPJ_KEY_REGION_CHANNEL_TABLE_SIZE</i>	34	[0,50]	0/1	–	R/W
<i>E_IPJ_KEY_REGION_ON_TIME_NOMINAL</i>	35	uint32	0/1	ms	R/W
<i>E_IPJ_KEY_REGION_ON_TIME_ACCESS</i>	36	uint32	0/1	ms	R/W
<i>E_IPJ_KEY_REGION_OFF_TIME</i>	37	uint32	0/1	ms	R/W
<i>E_IPJ_KEY_REGION_OFF_TIME_SAME_CHANNEL</i>	38	uint32	0/1	ms	R/W
<i>E_IPJ_KEY_REGION_START_FREQUENCY_KHZ</i>	39	uint32	0/1	kHz	R/W
<i>E_IPJ_KEY_REGION_CHANNEL_SPACING_KHZ</i>	40	uint32	0/1	kHz	R/W
<i>E_IPJ_KEY_REGION_RANDOM_HOP</i>	41	bool	0/1	–	R/W
<i>E_IPJ_KEY_REGION_INDU_PLL_R_DIVIDER</i>	42	[24,60]	0/1	–	R/W
<i>E_IPJ_KEY_REGION_RF_FILTER</i>	43	[0,2]	0/1	–	R/W

Report

Key	Key Id	Range/Type	Banks/Values	Units	R/W
<i>E_IPJ_KEY_REPORT_CONTROL_TAG</i>	161	<i>ipj_tag_flag</i>	0/1	–	R/W
<i>E_IPJ_KEY_REPORT_CONTROL_STATUS</i>	162	<i>ipj_status_flag</i>	0/1	–	R/W
<i>E_IPJ_KEY_REPORT_CONTROL_TIMESTAMP</i>	163	<i>ipj_report_timestamp_flag</i>	0/1	–	R/W
<i>E_IPJ_KEY_RESPONSE_CONTROL_TIMESTAMP</i>	164	<i>ipj_response_timestamp_flag</i>	0/1	–	R/W
<i>E_IPJ_KEY_ENABLE_LT_REPORTS</i>	259	bool	0/1	–	R/W

Select

Key	Key Id	Range/Type	Banks/Values	Units	R/W
<i>E_IPJ_KEY_SELECT_ENABLE</i>	80	bool	2/1	–	R/W
<i>E_IPJ_KEY_SELECT_TARGET</i>	81	<i>ipj_select_target</i>	2/1	–	R/W
<i>E_IPJ_KEY_SELECT_ACTION</i>	82	<i>ipj_select_action</i>	2/1	–	R/W
<i>E_IPJ_KEY_SELECT_MEM_BANK</i>	83	<i>ipj_mem_bank</i>	2/1	–	R/W
<i>E_IPJ_KEY_SELECT_POINTER</i>	84	int32	2/1	–	R/W
<i>E_IPJ_KEY_SELECT_MASK_LENGTH</i>	85	[0,255]	2/1	–	R/W
<i>E_IPJ_KEY_SELECT_MASK_VALUE</i>	86	uint16	2/16	–	R/W

Tag Access

Key	Key Id	Range/Type	Banks/Values	Units	R/W
<i>E_IPJ_KEY_TAG_OPERATION</i>	96	<i>ipj_tag_operation_type</i>	0/1	–	R/W
<i>E_IPJ_KEY_ACCESS_PASSWORD</i>	97	uint32	0/1	–	R/W
<i>E_IPJ_KEY_KILL_PASSWORD</i>	98	uint32	0/1	–	R/W
<i>E_IPJ_KEY_READ_MEM_BANK</i>	99	<i>ipj_mem_bank</i>	0/1	–	R/W
<i>E_IPJ_KEY_READ_WORD_POINTER</i>	100	uint32	0/1	–	R/W
<i>E_IPJ_KEY_READ_WORD_COUNT</i>	101	[0,32]	0/1	–	R/W
<i>E_IPJ_KEY_WRITE_MEM_BANK</i>	102	<i>ipj_mem_bank</i>	0/1	–	R/W
<i>E_IPJ_KEY_WRITE_WORD_POINTER</i>	103	uint32	0/1	–	R/W
<i>E_IPJ_KEY_WRITE_WORD_COUNT</i>	104	[0,32]	0/1	–	R/W
<i>E_IPJ_KEY_WRITE_DATA</i>	105	uint16	0/32	–	R/W
<i>E_IPJ_KEY_LOCK_PAYLOAD</i>	106	[0,0xFFFFF]	0/1	–	R/W
<i>E_IPJ_KEY_BLOCKPERMALOCK_ACTION</i>	107	<i>ipj_blockpermalock_action</i>	0/1	–	R/W
<i>E_IPJ_KEY_BLOCKPERMALOCK_MEM_BANK</i>	108	<i>ipj_mem_bank</i>	0/1	–	R/W
<i>E_IPJ_KEY_BLOCKPERMALOCK_BLOCK_POINTER</i>	109	uint32	0/1	–	R/W
<i>E_IPJ_KEY_BLOCKPERMALOCK_BLOCK_RANGE</i>	110	uint32	0/1	–	R/W
<i>E_IPJ_KEY_BLOCKPERMALOCK_MASK</i>	111	uint16	0/16	–	R/W
<i>E_IPJ_KEY_WRITE_EPC_LENGTH_CONTROL</i>	112	<i>ipj_write_epc_length_control</i>	0/1	–	R/W
<i>E_IPJ_KEY_WRITE_EPC_LENGTH_VALUE</i>	113	[0,31]	0/1	–	R/W
<i>E_IPJ_KEY_WRITE_EPC_AFI_CONTROL</i>	114	uint8	0/1	–	R/W
<i>E_IPJ_KEY_WRITE_EPC_AFI_VALUE</i>	115	uint8	0/1	–	R/W
<i>E_IPJ_KEY_QT_ACTION</i>	116	<i>ipj_qt_action</i>	0/1	–	R/W
<i>E_IPJ_KEY_QT_PERSISTENCE</i>	117	<i>ipj_qt_persistence</i>	0/1	–	R/W
<i>E_IPJ_KEY_QT_DATA_PROFILE</i>	118	<i>ipj_qt_data_profile</i>	0/1	–	R/W
<i>E_IPJ_KEY_QT_ACCESS_RANGE</i>	119	<i>ipj_qt_access_range</i>	0/1	–	R/W
<i>E_IPJ_KEY_QT_TAG_OPERATION</i>	120	<i>ipj_tag_operation_type</i>	0/1	–	R/W

Test

Key	Key Id	Range/Type	Banks/Values	Units	R/W
<i>E_IPJ_KEY_TEST_ID</i>	1024	uint32	0/1	–	R/W
<i>E_IPJ_KEY_TEST_PARAMETERS</i>	1025	uint32	0/16	–	R/W
<i>E_IPJ_KEY_TEST_RESULT_1</i>	1026	uint32	0/1	–	R
<i>E_IPJ_KEY_TEST_RESULT_2</i>	1027	uint32	0/1	–	R
<i>E_IPJ_KEY_TEST_RESULT_3</i>	1028	uint32	0/1	–	R
<i>E_IPJ_KEY_TEST_DATA</i>	1029	uint32	0/16	–	R
<i>E_IPJ_KEY_TEST_FREQUENCY</i>	1030	uint32	0/1	–	R
<i>E_IPJ_KEY_TEST_POWER</i>	1031	uint32	0/22	–	R
<i>E_IPJ_KEY_TEST_RF_MODE</i>	1032	uint32	0/5	–	R
<i>E_IPJ_KEY_TEST_TIME</i>	1033	uint32	0/1	–	R
<i>E_IPJ_KEY_TEST_EVENT</i>	1034	uint32	0/8	–	R
<i>E_IPJ_KEY_TEST_REPORTS</i>	1035	uint32	0/6	–	R
<i>E_IPJ_KEY_TEST_SYSTEM</i>	1036	uint32	0/1	–	R
<i>E_IPJ_KEY_TEST_DEBUG_PORT</i>	1037	uint32	0/1	–	R/W

Key Codes By Key Id

E_IPJ_KEY_BOOTSTRAP_VERSION

- Key Id: 1
- Permissions: R
- Range/Type: uint32
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: Dynamic
- Description: Bootstrap Version

E_IPJ_KEY_BOOTSTRAP_CRC

- Key Id: 2
- Permissions: R
- Range/Type: uint32
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: Dynamic
- Description: Bootstrap CRC

E_IPJ_KEY_APPLICATION_VERSION

- Key Id: 3
- Permissions: R
- Range/Type: uint32
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: Dynamic
- Description: Application Version

E_IPJ_KEY_APPLICATION_CRC

- Key Id: 4
- Permissions: R

- Range/Type: uint32
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: Dynamic
- Description: Application CRC

E_IPJ_KEY_SECONDARY_IMAGE_VERSION

- Key Id: 5
- Permissions: R
- Range/Type: uint32
- Units: –
- Bank Count: 3
- Value Count: 1
- Default: Dynamic
- Description: Secondary Image Version

E_IPJ_KEY_SECONDARY_IMAGE_CRC

- Key Id: 6
- Permissions: R
- Range/Type: uint32
- Units: –
- Bank Count: 3
- Value Count: 1
- Default: Dynamic
- Description: Secondary Image CRC

E_IPJ_KEY_SECONDARY_IMAGE_TYPE

- Key Id: 7
- Permissions: R
- Range/Type: uint32
- Units: –
- Bank Count: 3
- Value Count: 1
- Default: Dynamic

- Description: Secondary Image Type

E_IPJ_KEY_APPLICATION_REVISION_ID

- Key Id: 8
- Permissions: R
- Range/Type: uint32
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: Dynamic
- Description: Application Revision Id

E_IPJ_KEY_APPLICATION_BUILD_ID

- Key Id: 9
- Permissions: R
- Range/Type: uint32
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: Dynamic
- Description: Application Build Id

E_IPJ_KEY_PRODUCT_ID

- Key Id: 10
- Permissions: R
- Range/Type: *ipj_product_id*
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: Dynamic
- Description: Product Id

E_IPJ_KEY_SERIAL_NUMBER

- Key Id: 11
- Permissions: R
- Range/Type: uint32
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: Dynamic
- Description: Serial Number within a lot

E_IPJ_KEY_TRANSCEIVER_ID

- Key Id: 12
- Permissions: R
- Range/Type: uint32
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: Dynamic
- Description: Transceiver ID

E_IPJ_KEY_MICROPROCESSOR_ID

- Key Id: 13
- Permissions: R
- Range/Type: uint32
- Units: –
- Bank Count: 0
- Value Count: 4
- Default: Dynamic
- Description: Microcontroller ID and unique identifier

E_IPJ_KEY_CUSTOMER_VERSION

- Key Id: 14
- Permissions: R
- Range/Type: uint32
- Units: –

- Bank Count: 0
- Value Count: 1
- Default: Dynamic
- Description: Customer Version

E_IPJ_KEY_CUSTOMER_ID

- Key Id: 15
- Permissions: R
- Range/Type: uint32
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: Dynamic
- Description: Customer ID

E_IPJ_KEY_CUSTOMER_PRODUCT_ID

- Key Id: 16
- Permissions: R
- Range/Type: uint32
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: Dynamic
- Description: Customer Product ID

E_IPJ_KEY_CALIBRATION_INFO

- Key Id: 17
- Permissions: R
- Range/Type: uint32
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: Dynamic
- Description: Calibration Source Information

E_IPJ_KEY_TEST_INFO

- Key Id: 18
- Permissions: R
- Range/Type: uint32
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: The number of test commands executed

E_IPJ_KEY_PRODUCT_SKU

- Key Id: 19
- Permissions: R
- Range/Type: *ipj_product_sku*
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: Dynamic
- Description: The hardware SKU of the reader

E_IPJ_KEY_LOT_DATE_CODE

- Key Id: 20
- Permissions: R
- Range/Type: uint32
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: Dynamic
- Description: Combined lot and date code ZZWWYY where ZZ is lot number, WW is work week, and YY is year produced

E_IPJ_KEY_PRODUCT_KEY

- Key Id: 21
- Permissions: R
- Range/Type: uint32

- Units: –
- Bank Count: 0
- Value Count: 1
- Default: Dynamic
- Description: Product Key

E_IPJ_KEY_SECONDARY_IMAGE_LOCATION

- Key Id: 22
- Permissions: R
- Range/Type: uint32
- Units: –
- Bank Count: 3
- Value Count: 1
- Default: Dynamic
- Description: Secondary Image Location

E_IPJ_KEY_SECONDARY_IMAGE_SIZE

- Key Id: 23
- Permissions: R
- Range/Type: uint32
- Units: –
- Bank Count: 3
- Value Count: 1
- Default: Dynamic
- Description: Secondary Image Size

E_IPJ_KEY_UNIQUE_ID

- Key Id: 24
- Permissions: R
- Range/Type: uint32
- Units: –
- Bank Count: 0
- Value Count: 2
- Default: Dynamic
- Description: 64-bit Unique Id for each device XXZZWWYYAAAA where XX is SKU, ZZWWYY is Lot Date Code, and AAAA is Serial Number within the lot

E_IPJ_KEY_HARDWARE_REVISION

- Key Id: 25
- Permissions: R
- Range/Type: uint8
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: Dynamic
- Description: The hardware revision of the reader.

E_IPJ_KEY_REGION_ID

- Key Id: 32
- Permissions: R/W
- Range/Type: *ipj_region*
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: Dynamic
- Description: Region ID. Please refer to section Regulatory Region for the list of supported regions.

E_IPJ_KEY_REGION_CHANNEL_TABLE

- Key Id: 33
- Permissions: R/W
- Range/Type: uint8
- Units: –
- Bank Count: 0
- Value Count: 50
- Default: Dynamic
- Description: Channel Table. If this key code is not configured, the default channel table for the current regulatory region is used. The mapping from channel indices to frequency values depends upon the regulatory region.

E_IPJ_KEY_REGION_CHANNEL_TABLE_SIZE

- Key Id: 34
- Permissions: R/W
- Range/Type: [0,50]

- Units: –
- Bank Count: 0
- Value Count: 1
- Default: Dynamic
- Description: Channel Table usable entry size

E_IPJ_KEY_REGION_ON_TIME_NOMINAL

- Key Id: 35
- Permissions: R/W
- Range/Type: uint32
- Units: ms
- Bank Count: 0
- Value Count: 1
- Default: Dynamic
- Description: Dwell time (in ms). Only valid for user-specified region. Recommended that this be 200ms less than REGION_ON_TIME_ACCESS

E_IPJ_KEY_REGION_ON_TIME_ACCESS

- Key Id: 36
- Permissions: R/W
- Range/Type: uint32
- Units: ms
- Bank Count: 0
- Value Count: 1
- Default: Dynamic
- Description: Dwell time (in ms) when access operation performed. Only valid for user-specified region.

E_IPJ_KEY_REGION_OFF_TIME

- Key Id: 37
- Permissions: R/W
- Range/Type: uint32
- Units: ms
- Bank Count: 0
- Value Count: 1
- Default: Dynamic
- Description: Off-time (in ms) when switching to a different channel. Only valid for user-specified region.

E_IPJ_KEY_REGION_OFF_TIME_SAME_CHANNEL

- Key Id: 38
- Permissions: R/W
- Range/Type: uint32
- Units: ms
- Bank Count: 0
- Value Count: 1
- Default: Dynamic
- Description: Off-time (in ms) when switching to the same channel. Only valid for user-specified region.

E_IPJ_KEY_REGION_START_FREQUENCY_KHZ

- Key Id: 39
- Permissions: R/W
- Range/Type: uint32
- Units: kHz
- Bank Count: 0
- Value Count: 1
- Default: Dynamic
- Description: Channel 1 frequency in kHz. Only valid for user-specified region.

E_IPJ_KEY_REGION_CHANNEL_SPACING_KHZ

- Key Id: 40
- Permissions: R/W
- Range/Type: uint32
- Units: kHz
- Bank Count: 0
- Value Count: 1
- Default: Dynamic
- Description: Channel spacing in kHz. Only valid for user-specified region.

E_IPJ_KEY_REGION_RANDOM_HOP

- Key Id: 41
- Permissions: R/W
- Range/Type: bool
- Units: –

- Bank Count: 0
- Value Count: 1
- Default: Dynamic
- **Description: Specifies a random hop sequence. Only valid for user-specified region.**
 - 0: Hopping off
 - 1: Hopping on

E_IPJ_KEY_REGION_INDY_PLL_R_DIVIDER

- Key Id: 42
- Permissions: R/W
- Range/Type: [24,60]
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: Dynamic
- Description: Indy PLL parameter. Recommended values are 24, 30, 48, and 60. Only valid for user-specified region.

E_IPJ_KEY_REGION_RF_FILTER

- Key Id: 43
- Permissions: R/W
- Range/Type: [0,2]
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: Dynamic
- **Description: External RF Filter selection.**
 - 0 = FCC
 - 1 = EU
 - 2 = JP

E_IPJ_KEY_ANTENNA_TX_POWER

- Key Id: 49
- Permissions: R/W
- Range/Type: [0,3150]
- Units: cdBm

- Bank Count: 1
- Value Count: 1
- Default: Dynamic
- Description: Antenna Transmit Power in cdBm (eg. 2300 = 23 dBm). By default, this is set to the maximum rated power of the device and region combination. For example, the RS2000 configured for the USA defaults to 31.5 dBm transmit power.

E_IPJ_KEY_ANTENNA_SEQUENCE

- Key Id: 50
- Permissions: R/W
- Range/Type: uint8
- Units: –
- Bank Count: 0
- Value Count: 16
- Default: 0
- Description: The sequence of antennas through which the reader will cycle. Each value represents one antenna index in the sequence. Any values of 0 will be skipped.

E_IPJ_KEY_INVENTORY_TAG_POPULATION

- Key Id: 64
- Permissions: R/W
- Range/Type: uint32
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 16
- Description: An estimate of the tag population in view of the RF field of the antenna.

E_IPJ_KEY_INVENTORY_SELECT_FLAG

- Key Id: 65
- Permissions: R/W
- Range/Type: *ipj_inventory_select_flag*
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 1

- Description: Inventory Select Flag. Determines which Tags will respond during Inventory.

E_IPJ_KEY_INVENTORY_SESSION

- Key Id: 66
- Permissions: R/W
- Range/Type: [0,3]
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: Inventory Session Number (0 - 3)

E_IPJ_KEY_INVENTORY_SEARCH_MODE

- Key Id: 67
- Permissions: R/W
- Range/Type: *ipj_inventory_search_mode*
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: Inventory Search Mode

E_IPJ_KEY_FAST_ID_ENABLE

- Key Id: 69
- Permissions: R/W
- Range/Type: bool
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: Enable FastID Capability for Monza Tags

E_IPJ_KEY_TAG_FOCUS_ENABLE

- Key Id: 70
- Permissions: R/W
- Range/Type: bool
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: Tag Focus Capability for Monza Tags (Session must be S1 and Search Mode must be A->B Only)

E_IPJ_KEY_TAG_OPERATION_ENABLE

- Key Id: 71
- Permissions: R/W
- Range/Type: bool
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: Tag Operation Enable during inventory

E_IPJ_KEY_TAG_OPERATION_RETRIES

- Key Id: 72
- Permissions: R/W
- Range/Type: uint8
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 3
- Description: Maximum number of retries on a failed tag access operation. If a valid tag response with a Memory Locked or Memory Overrun Gen2 Error is received, the tag access operation is not retried.

E_IPJ_KEY_SELECT_ENABLE

- Key Id: 80
- Permissions: R/W
- Range/Type: bool

- Units: –
- Bank Count: 2
- Value Count: 1
- Default: 0
- Description: Enables the transmission of a Select command before an inventory round.

E_IPJ_KEY_SELECT_TARGET

- Key Id: 81
- Permissions: R/W
- Range/Type: *ipj_select_target*
- Units: –
- Bank Count: 2
- Value Count: 1
- Default: 0
- Description: Determines if the Select command modifies a Tag's SL flag or its inventoried flag. In the case of the inventoried flag it further determines one of the four sessions

E_IPJ_KEY_SELECT_ACTION

- Key Id: 82
- Permissions: R/W
- Range/Type: *ipj_select_action*
- Units: –
- Bank Count: 2
- Value Count: 1
- Default: 0
- Description: Specifies if during the Select command, matching tags should assert SL, de-assert SL, or set their inventoried flag to A or to B.

E_IPJ_KEY_SELECT_MEM_BANK

- Key Id: 83
- Permissions: R/W
- Range/Type: *ipj_mem_bank*
- Units: –
- Bank Count: 2
- Value Count: 1
- Default: 0

- Description: The memory bank on which the tag filter is applied. Tag filters may be configured to search for content in the Epc, Tid, and User memory banks. Tag filters will not match against the Reserved memory bank.

E_IPJ_KEY_SELECT_POINTER

- Key Id: 84
- Permissions: R/W
- Range/Type: int32
- Units: –
- Bank Count: 2
- Value Count: 1
- Default: 0
- Description: The bit offset in the specified memory bank at which the tag mask begins. This is bit offset and need not be word or even byte-aligned.

E_IPJ_KEY_SELECT_MASK_LENGTH

- Key Id: 85
- Permissions: R/W
- Range/Type: [0,255]
- Units: –
- Bank Count: 2
- Value Count: 1
- Default: 0
- Description: This key code along with SELECT_POINTER key determine the memory range over which the Select command mask (SELECT_MASK_VALUE) is applied. This key code specifies the length of the memory range in bits.

E_IPJ_KEY_SELECT_MASK_VALUE

- Key Id: 86
- Permissions: R/W
- Range/Type: uint16
- Units: –
- Bank Count: 2
- Value Count: 16
- Default: 0
- Description: The tag mask defines the bit pattern that the tag filter must match on. For a non-16 bit aligned tag mask the final bits are left justified (high order bits) in the last word.

E_IPJ_KEY_TAG_OPERATION

- Key Id: 96
- Permissions: R/W
- Range/Type: *ipj_tag_operation_type*
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: Determines the specific Access command issued to a Tag when Tag Operation enabled.

E_IPJ_KEY_ACCESS_PASSWORD

- Key Id: 97
- Permissions: R/W
- Range/Type: uint32
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: Specifies the 32-bit Access password that is used in conjunction with the EPCglobal Gen2 Access command to move the Tag to the Secured state. If the key value is non-zero, the reader implements the access procedure before issuing access commands.

E_IPJ_KEY_KILL_PASSWORD

- Key Id: 98
- Permissions: R/W
- Range/Type: uint32
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: Specifies the 32-bit Kill password used to kill the Tag.

E_IPJ_KEY_READ_MEM_BANK

- Key Id: 99
- Permissions: R/W

- Range/Type: *ipj_mem_bank*
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: Memory Bank to access for read operation.

E_IPJ_KEY_READ_WORD_POINTER

- Key Id: 100
- Permissions: R/W
- Range/Type: uint32
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: Word Pointer to access for read operation.

E_IPJ_KEY_READ_WORD_COUNT

- Key Id: 101
- Permissions: R/W
- Range/Type: [0,32]
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: Number of words to access for read operation.

E_IPJ_KEY_WRITE_MEM_BANK

- Key Id: 102
- Permissions: R/W
- Range/Type: *ipj_mem_bank*
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0

- Description: Memory Bank to access for write operation.

`E_IPJ_KEY_WRITE_WORD_POINTER`

- Key Id: 103
- Permissions: R/W
- Range/Type: uint32
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: Word Pointer to access for write operation.

`E_IPJ_KEY_WRITE_WORD_COUNT`

- Key Id: 104
- Permissions: R/W
- Range/Type: [0,32]
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: Number of words to write.

`E_IPJ_KEY_WRITE_DATA`

- Key Id: 105
- Permissions: R/W
- Range/Type: uint16
- Units: –
- Bank Count: 0
- Value Count: 32
- Default: 0
- Description: Data to write into the tag memory. Specified in 16-bit words.

E_IPJ_KEY_LOCK_PAYLOAD

- Key Id: 106
- Permissions: R/W
- Range/Type: [0,0xFFFFF]
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: Payload field for the lock command. Specified in a 20-bit value.

E_IPJ_KEY_BLOCKPERMALOCK_ACTION

- Key Id: 107
- Permissions: R/W
- Range/Type: *ipj_blockpermalock_action*
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: Blockpermalock action for blockpermalock operation.

E_IPJ_KEY_BLOCKPERMALOCK_MEM_BANK

- Key Id: 108
- Permissions: R/W
- Range/Type: *ipj_mem_bank*
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: Memory Bank to access for blockpermalock operation.

E_IPJ_KEY_BLOCKPERMALOCK_BLOCK_POINTER

- Key Id: 109
- Permissions: R/W
- Range/Type: uint32
- Units: –

- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: Block Pointer for the blockpermalock operation.

E_IPJ_KEY_BLOCKPERMALOCK_BLOCK_RANGE

- Key Id: 110
- Permissions: R/W
- Range/Type: uint32
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: Blockpermalock mask range in units of 16 blocks for the blockpermalock operation.

E_IPJ_KEY_BLOCKPERMALOCK_MASK

- Key Id: 111
- Permissions: R/W
- Range/Type: uint16
- Units: –
- Bank Count: 0
- Value Count: 16
- Default: 0
- Description: Blockpermalock mask for the blockpermalock operation.

E_IPJ_KEY_WRITE_EPC_LENGTH_CONTROL

- Key Id: 112
- Permissions: R/W
- Range/Type: *ipj_write_epc_length_control*
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: EPC length handling control for the WRITE_EPC tag operation. There are options to automatically update the length, specify a user length, zero the length, or do not change the length value.

E_IPJ_KEY_WRITE_EPC_LENGTH_VALUE

- Key Id: 113
- Permissions: R/W
- Range/Type: [0,31]
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: The user specified EPC length value when user value EPC length control is selected.

E_IPJ_KEY_WRITE_EPC_AFI_CONTROL

- Key Id: 114
- Permissions: R/W
- Range/Type: uint8
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: This enables the write of the AFI bits in the tag EPC memory if set to True. AFI bits are bits 18h-1Fh in EPC memory. The AFI bits will be updated to the user value set by E_IPJ_KEY_WRITE_EPC_AFI_VALUE. Please note that bit 17h in EPC memory should have a logical 1 for bits 18h-1Fh to contain a valid AFI value.

E_IPJ_KEY_WRITE_EPC_AFI_VALUE

- Key Id: 115
- Permissions: R/W
- Range/Type: uint8
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: The user specified AFI Bits when AFI Control is enabled.

E_IPJ_KEY_QT_ACTION

- Key Id: 116
- Permissions: R/W
- Range/Type: *ipj_qt_action*
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: QT Action for the QT operation.

E_IPJ_KEY_QT_PERSISTENCE

- Key Id: 117
- Permissions: R/W
- Range/Type: *ipj_qt_persistence*
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: QT Persistence for the QT operation.

E_IPJ_KEY_QT_DATA_PROFILE

- Key Id: 118
- Permissions: R/W
- Range/Type: *ipj_qt_data_profile*
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: QT Data Profile for the QT operation.

E_IPJ_KEY_QT_ACCESS_RANGE

- Key Id: 119
- Permissions: R/W
- Range/Type: *ipj_qt_access_range*
- Units: –

- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: QT Access Range for the QT operation.

E_IPJ_KEY_QT_TAG_OPERATION

- Key Id: 120
- Permissions: R/W
- Range/Type: *ipj_tag_operation_type*
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: Determines the specific Access command issued to a Tag when Tag Operation enabled after the QT Command. Only Read and Write are supported.

E_IPJ_KEY_AUTOSTOP_DURATION_MS

- Key Id: 137
- Permissions: R/W
- Range/Type: uint32
- Units: ms
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: Specifies the duration of time to Inventory. Inventory stops automatically once the specified time has elapsed.

E_IPJ_KEY_AUTOSTOP_TAG_COUNT

- Key Id: 139
- Permissions: R/W
- Range/Type: uint32
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0

- Description: Specifies the number of Tags to Inventory. Inventory stops automatically once the specified number of Tags are inventoried.

E_IPJ_KEY_REPORT_CONTROL_TAG

- Key Id: 161
- Permissions: R/W
- Range/Type: *ipj_tag_flag*
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0x00000023
- Description: Controls which Tag fields are present in a TagOperationReport

E_IPJ_KEY_REPORT_CONTROL_STATUS

- Key Id: 162
- Permissions: R/W
- Range/Type: *ipj_status_flag*
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: Controls which Status reports will be generated

E_IPJ_KEY_REPORT_CONTROL_TIMESTAMP

- Key Id: 163
- Permissions: R/W
- Range/Type: *ipj_report_timestamp_flag*
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: Controls which Reports will contain timestamps

E_IPJ_KEY_RESPONSE_CONTROL_TIMESTAMP

- Key Id: 164
- Permissions: R/W
- Range/Type: *ipj_response_timestamp_flag*
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: Controls which Responses will contain timestamps

E_IPJ_KEY_GPIO_MODE

- Key Id: 192
- Permissions: R/W
- Range/Type: *ipj_gpio_mode*
- Units: –
- Bank Count: 5
- Value Count: 1
- Default: 0
- Description: Controls the GPIO mode (Input/output/action/etc)

E_IPJ_KEY_GPIO_STATE

- Key Id: 193
- Permissions: R/W
- Range/Type: *ipj_gpio_state*
- Units: –
- Bank Count: 5
- Value Count: 1
- Default: 0
- Description: Controls the GPIO Logic level (+3.3 V/0.0 V). Output is driven (maximum load is +/- 8 mA), input is pulled internally via resistors(or left floating).

E_IPJ_KEY_GPIO_HI_ACTION

- Key Id: 194
- Permissions: R/W
- Range/Type: *ipj_gpi_action*

- Units: –
- Bank Count: 5
- Value Count: 1
- Default: 0
- Description: Control the action when GPIO transitions High. Note that this MUST NOT be set to the same value as GPIO_LO_ACTION on a given pin (undefined behavior will result)

E_IPJ_KEY_GPIO_LO_ACTION

- Key Id: 195
- Permissions: R/W
- Range/Type: *ipj_gpi_action*
- Units: –
- Bank Count: 5
- Value Count: 1
- Default: 0
- Description: Controls the action when GPIO transitions Low. Note that this MUST NOT be set to the same value as GPIO_HI_ACTION on a given pin (undefined behavior will result)

E_IPJ_KEY_GPIO_DEBOUNCE_MS

- Key Id: 197
- Permissions: R/W
- Range/Type: uint32
- Units: ms
- Bank Count: 5
- Value Count: 1
- Default: 0
- Description: Controls internal debounce timeout for GPI actions

E_IPJ_KEY_GPIO_CURRENT_STATE

- Key Id: 198
- Permissions: R
- Range/Type: *ipj_gpio_state*
- Units: –
- Bank Count: 5
- Value Count: 1
- Default: 0

- Description: Reflects the current logic level (+3.3 V/0.0 V) of the GPIO pins. State is updated while running GPIO action

E_IPJ_KEY_RF_MODE

- Key Id: 208
- Permissions: R/W
- Range/Type: [0,4]
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- **Description: RF Mode for inventory operation:**
 - Mode 0: Auto (Default Mode 1)
 - Mode 1: 25 us Tari, M4, 250 kHz
 - Mode 2: 25 us Tari, M4, 300 kHz
 - Mode 3: 6.25 us Tari, FM0, 400 kHz
 - Mode 4: 25 us Tari, FM0, 40 kHz

E_IPJ_KEY_FIRST_ERROR

- Key Id: 224
- Permissions: R
- Range/Type: *ipj_error*
- Units: –
- Bank Count: 0
- Value Count: 5
- Default: 0
- Description: Holds the first error that occurred since last cleared (or boot)

E_IPJ_KEY_LAST_ERROR

- Key Id: 225
- Permissions: R
- Range/Type: *ipj_error*
- Units: –
- Bank Count: 0
- Value Count: 5
- Default: 0

- Description: Holds the Last error that occurred

E_IPJ_KEY_SYSTEM_ERROR

- Key Id: 226
- Permissions: R
- Range/Type: *ipj_error*
- Units: –
- Bank Count: 0
- Value Count: 5
- Default: 0
- Description: Holds the last System level error that occurred (Hard Faults)

E_IPJ_KEY_DEVICE_BAUDRATE

- Key Id: 256
- Permissions: R/W
- Range/Type: *ipj_baud_rate*
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: Serial baud rate the device connects at after boot

E_IPJ_KEY_DEVICE_IDLE_POWER_MODE

- Key Id: 257
- Permissions: R/W
- Range/Type: *ipj_idle_power_mode*
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: Power consumption mode when the device is idle

E_IPJ_KEY_ONBOOT_START_ACTION

- Key Id: 258
- Permissions: R/W
- Range/Type: *ipj_action*
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: E_IPJ_ACTION_NONE
- Description: If an action is set in the device stored settings then the action will automatically start on device boot.

E_IPJ_KEY_ENABLE_LT_REPORTS

- Key Id: 259
- Permissions: R/W
- Range/Type: bool
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: Useful in an RX only use case for the ITK-LT host library. The reader can be configured with an ONBOOT_START_ACTION and this key to automatically send up LT tag operation reports.

E_IPJ_KEY_TEST_ID

- Key Id: 1024
- Permissions: R/W
- Range/Type: uint32
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: (ENGINEERING USE ONLY - SUBJECT TO CHANGE) Test Command Id is used to select the specific test command. Please refer to ITK examples for usage.

E_IPJ_KEY_TEST_PARAMETERS

- Key Id: 1025
- Permissions: R/W
- Range/Type: uint32
- Units: –
- Bank Count: 0
- Value Count: 16
- Default: 0
- Description: (ENGINEERING USE ONLY - SUBJECT TO CHANGE) Test Command Parameters provide inputs for specific test commands. Please refer to ITK examples for usage.

E_IPJ_KEY_TEST_RESULT_1

- Key Id: 1026
- Permissions: R
- Range/Type: uint32
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: (ENGINEERING USE ONLY - SUBJECT TO CHANGE) Test Command Result 1

E_IPJ_KEY_TEST_RESULT_2

- Key Id: 1027
- Permissions: R
- Range/Type: uint32
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: (ENGINEERING USE ONLY - SUBJECT TO CHANGE) Test Command Result 2

E_IPJ_KEY_TEST_RESULT_3

- Key Id: 1028
- Permissions: R
- Range/Type: uint32

- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: (ENGINEERING USE ONLY - SUBJECT TO CHANGE) Test Command Result 3

E_IPJ_KEY_TEST_DATA

- Key Id: 1029
- Permissions: R
- Range/Type: uint32
- Units: –
- Bank Count: 0
- Value Count: 16
- Default: 0
- Description: (ENGINEERING USE ONLY - SUBJECT TO CHANGE) Test Command Data

E_IPJ_KEY_TEST_FREQUENCY

- Key Id: 1030
- Permissions: R
- Range/Type: uint32
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: (ENGINEERING USE ONLY - SUBJECT TO CHANGE) Last locked frequency

E_IPJ_KEY_TEST_POWER

- Key Id: 1031
- Permissions: R
- Range/Type: uint32
- Units: –
- Bank Count: 0
- Value Count: 22
- Default: 0
- Description: (ENGINEERING USE ONLY - SUBJECT TO CHANGE) Last transmit power

E_IPJ_KEY_TEST_RF_MODE

- Key Id: 1032
- Permissions: R
- Range/Type: uint32
- Units: –
- Bank Count: 0
- Value Count: 5
- Default: 0
- Description: (ENGINEERING USE ONLY - SUBJECT TO CHANGE) RF Profile identifier

E_IPJ_KEY_TEST_TIME

- Key Id: 1033
- Permissions: R
- Range/Type: uint32
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: (ENGINEERING USE ONLY - SUBJECT TO CHANGE) Last transmit on time

E_IPJ_KEY_TEST_EVENT

- Key Id: 1034
- Permissions: R
- Range/Type: uint32
- Units: –
- Bank Count: 0
- Value Count: 8
- Default: 0
- Description: (ENGINEERING USE ONLY - SUBJECT TO CHANGE) Event Info

E_IPJ_KEY_TEST_REPORTS

- Key Id: 1035
- Permissions: R
- Range/Type: uint32
- Units: –

- Bank Count: 0
- Value Count: 6
- Default: 0
- Description: (ENGINEERING USE ONLY - SUBJECT TO CHANGE) Report Info

E_IPJ_KEY_TEST_SYSTEM

- Key Id: 1036
- Permissions: R
- Range/Type: uint32
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: (ENGINEERING USE ONLY - SUBJECT TO CHANGE) System Info

E_IPJ_KEY_TEST_DEBUG_PORT

- Key Id: 1037
- Permissions: R/W
- Range/Type: uint32
- Units: –
- Bank Count: 0
- Value Count: 1
- Default: 0
- Description: (ENGINEERING USE ONLY - SUBJECT TO CHANGE) Serial Debug Port Configuration

E_IPJ_KEY_GENERIC_DATA

- Key Id: 3072
- Permissions: R/W
- Range/Type: uint32
- Units: –
- Bank Count: 1
- Value Count: 16
- Default: 0
- Description: Generic data storage for third parties to add custom data to the device.

E_IPJ_KEY_OEM_DATA

- Key Id: 3073
- Permissions: R
- Range/Type: uint32
- Units: –
- Bank Count: 0
- Value Count: 16
- Default: 0
- Description: OEM data storage to add custom data to the device during calibration.

1.5.12 Regulatory Regions

This section describes regions supported by the Indy SiPs. For information on how to configure an Indy SiP for a specific region, see the configuration example *Regulatory Region*.

Countries

Table 1.11: Country List

Country	Standard	IRI Define
Argentina	<i>FCC Part 15.247</i>	<i>E_IPJ_REGION_FCC_PART_15_247</i>
Armenia	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
Australia (920-926)	<i>Australia 920-926 MHz</i>	<i>E_IPJ_REGION_AUSTRALIA_920_926_MHZ</i>
Austria	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
Azerbaijan	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
Belgium	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
Bosnia and Herzegovina	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
Brazil (902-907)	<i>Brazil 902-907 and 915-928 MHz</i>	<i>E_IPJ_REGION_BRAZIL_902_907_AND_915_928_MHZ</i>
Brazil (915-928)	<i>Brazil 902-907 and 915-928 MHz</i>	<i>E_IPJ_REGION_BRAZIL_902_907_AND_915_928_MHZ</i>
Bulgaria	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
Canada	<i>FCC Part 15.247</i>	<i>E_IPJ_REGION_FCC_PART_15_247</i>
Chile	<i>FCC Part 15.247</i>	<i>E_IPJ_REGION_FCC_PART_15_247</i>
China (920-925)	<i>China 920-925 MHz</i>	<i>E_IPJ_REGION_CHINA_920_925_MHZ</i>
Colombia	<i>FCC Part 15.247</i>	<i>E_IPJ_REGION_FCC_PART_15_247</i>
Costa Rica	<i>FCC Part 15.247</i>	<i>E_IPJ_REGION_FCC_PART_15_247</i>
Croatia	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
Cyprus	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
Czech Republic	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
Denmark	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
Dominican Republic	<i>FCC Part 15.247</i>	<i>E_IPJ_REGION_FCC_PART_15_247</i>
Estonia	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
Finland	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
France	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
Germany	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
Greece	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>

Continued on next page

Table 1.11 – continued from previous page

Country	Standard	IRI Define
Hong Kong (920-925)	<i>Hong Kong 920-925 MHz</i>	<i>E_IPJ_REGION_HONG_KONG_920_925_MHZ</i>
Hungary	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
Iceland	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
India	<i>India 865-867 MHz</i>	<i>E_IPJ_REGION_INDIA_865_867_MHZ</i>
Indonesia	<i>Indonesia 923-925 MHz</i>	<i>E_IPJ_REGION_INDONESIA_923_925_MHZ</i>
Ireland	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
Israel	<i>Israel 915-917 MHz</i>	<i>E_IPJ_REGION_ISRAEL_915_917_MHZ</i>
Italy	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
Japan (916-921)	<i>Japan 916-921 MHz, no LBT</i>	<i>E_IPJ_REGION_JAPAN_916_921_MHZ_NO_LBT</i>
Korea (917-921)	<i>Korea 917-920.8 MHz</i>	<i>E_IPJ_REGION_KOREA_917_921_MHZ</i>
Latvia	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
Lithuania	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
Luxembourg	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
Macedonia	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
Malaysia (919-923)	<i>Malaysia 919-923 MHz</i>	<i>E_IPJ_REGION_MALAYSIA_919_923_MHZ</i>
Malta	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
Mexico	<i>FCC Part 15.247</i>	<i>E_IPJ_REGION_FCC_PART_15_247</i>
Moldova	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
Netherlands	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
New Zealand (921-928)	<i>New Zealand 921.5-928 MHz</i>	<i>E_IPJ_REGION_NEW_ZEALAND_921P5_928_MHZ</i>
Norway	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
Oman	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
Panama	<i>FCC Part 15.247</i>	<i>E_IPJ_REGION_FCC_PART_15_247</i>
Peru	<i>Peru 916-928 MHz</i>	<i>E_IPJ_REGION_PERU_916_928_MHZ</i>
Philippines	<i>Philippines 918-920 MHz</i>	<i>E_IPJ_REGION_PHILIPPINES_918_920_MHZ</i>
Poland	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
Portugal	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
Romania	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
Russian Federation (916-921)	<i>Russia 916-921 MHz</i>	<i>E_IPJ_REGION_RUSSIA_916_921_MHZ</i>
Saudi Arabia	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
Serbia	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
Singapore (920-925)	<i>Singapore 920-925 MHz</i>	<i>E_IPJ_REGION_SINGAPORE_920_925_MHZ</i>
Slovak Republic	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
Slovenia	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
South Africa (915-919)	<i>South Africa 915-919 MHz</i>	<i>E_IPJ_REGION_SOUTH_AFRICA_915_919_MHZ</i>
Spain	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
Sweden	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
Switzerland	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
Taiwan (922-928)	<i>Taiwan 922-928 MHz</i>	<i>E_IPJ_REGION_TAIWAN_922_928_MHZ</i>
Thailand	<i>Thailand 920-925 MHz</i>	<i>E_IPJ_REGION_THAILAND_920_925_MHZ</i>
Turkey	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
United Arab Emirates	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
United Kingdom	<i>ETSI EN 302-208-1 V1.4.1</i>	<i>E_IPJ_REGION_ETSI_EN_302_208_V1_4_1</i>
United States	<i>FCC Part 15.247</i>	<i>E_IPJ_REGION_FCC_PART_15_247</i>
Uruguay	<i>Uruguay 916-928 MHz</i>	<i>E_IPJ_REGION_URUGUAY_916_928_MHZ</i>
Venezuela	<i>FCC Part 15.247</i>	<i>E_IPJ_REGION_FCC_PART_15_247</i>
Vietnam (920-925)	<i>Vietnam 920-925 MHz</i>	<i>E_IPJ_REGION_VIETNAM_920_925_MHZ</i>

Standards

FCC Part 15.247

The reader will operate in a random sequence over the frequencies specified in the channel table. The channel table is not configurable in this region.

The dwell time per channel is less than 200ms. The reader may dwell up to a maximum of 400ms while performing Tag Access operations. The following table lists the available channels for this region.

Table 1.12: Supported Channel List

Channel	Frequency (MHz)
1	902.75
2	903.25
3	903.75
4	904.25
5	904.75
6	905.25
7	905.75
8	906.25
9	906.75
10	907.25
11	907.75
12	908.25
13	908.75
14	909.25
15	909.75
16	910.25
17	910.75
18	911.25
19	911.75
20	912.25
21	912.75
22	913.25
23	913.75
24	914.25
25	914.75
26	915.25
27	915.75
28	916.25
29	916.75
30	917.25
31	917.75
32	918.25
33	918.75
34	919.25
35	919.75
36	920.25
37	920.75
38	921.25
39	921.75
Continued on next page	

Table 1.12 – continued from previous page

Channel	Frequency (MHz)
40	922.25
41	922.75
42	923.25
43	923.75
44	924.25
45	924.75
46	925.25
47	925.75
48	926.25
49	926.75
50	927.25

The Impinj RFID Reader will use all supported channels by default.

Supported List of Countries: Argentina, Canada, Chile, Colombia, Costa Rica, Dominican Republic, Mexico, Panama, United States, Venezuela

Hong Kong 920-925 MHz

The reader will operate in a random sequence over the frequencies specified in the channel table. The channel table is not configurable in this region.

The dwell time per channel is less than 200ms. The reader may dwell up to a maximum of 400ms while performing Tag Access operations. The following table lists the available channels for this region.

Table 1.13: Supported Channel List

Channel	Frequency (MHz)
1	920.25
2	920.75
3	921.25
4	921.75
5	922.25
6	922.75
7	923.25
8	923.75
9	924.25
10	924.75

The Impinj RFID Reader will use all supported channels by default.

Supported List of Countries: Hong Kong (920-925)

Taiwan 922-928 MHz

The reader will operate in a random sequence over the frequencies specified in the channel table. The channel table is not configurable in this region.

The dwell time per channel is less than 200ms. The reader may dwell up to a maximum of 400ms while performing Tag Access operations. The following table lists the available channels for this region.

Table 1.14: Supported Channel List

Channel	Frequency (MHz)
1	922.25
2	922.75
3	923.25
4	923.75
5	924.25
6	924.75
7	925.25
8	925.75
9	926.25
10	926.75
11	927.25
12	927.75

The Impinj RFID Reader will use all supported channels by default.

Supported List of Countries: Taiwan (922-928)

ETSI EN 302-208-1 V1.4.1

The reader will operate in a sequential sequence over the frequencies specified in the channel table. The channel table is configurable in this region.

If only 1 channel is specified in the channel table, the reader switches off for 100ms after a dwell time of 4000ms on the channel.

The reader turns off or switches channels if no tags are seen on a channel for 1000ms.

The dwell time per channel is less than 3800ms. The reader may dwell up to a maximum of 4000ms while performing Tag Access operations. The following table lists the available channels for this region.

Table 1.15: Supported Channel List

Channel	Frequency (MHz)
4	865.7
7	866.3
10	866.9
13	867.5

The Impinj RFID Reader will use all supported channels by default.

Supported List of Countries: Armenia, Austria, Azerbaijan, Belgium, Bosnia and Herzegovina, Bulgaria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Macedonia, Malta, Moldova, Netherlands, Norway, Oman, Poland, Portugal, Romania, Saudi Arabia, Serbia, Slovak Republic, Slovenia, Spain, Sweden, Switzerland, Turkey, United Arab Emirates, United Kingdom

Korea 917-920.8 MHz

The reader will operate in a random sequence over the frequencies specified in the channel table. The channel table is not configurable in this region.

The dwell time per channel is less than 200ms. The reader may dwell up to a maximum of 400ms while performing Tag Access operations. The following table lists the available channels for this region.

Table 1.16: Supported Channel List

Channel	Frequency (MHz)
1	917.3
2	917.9
3	918.5
4	919.1
5	919.7
6	920.3

The Impinj RFID Reader will use all supported channels by default.

Supported List of Countries: Korea (917-921)

Malaysia 919-923 MHz

The reader will operate in a random sequence over the frequencies specified in the channel table. The channel table is not configurable in this region.

The dwell time per channel is less than 200ms. The reader may dwell up to a maximum of 400ms while performing Tag Access operations. The following table lists the available channels for this region.

Table 1.17: Supported Channel List

Channel	Frequency (MHz)
1	919.25
2	919.75
3	920.25
4	920.75
5	921.25
6	921.75
7	922.25
8	922.75

The Impinj RFID Reader will use all supported channels by default.

Supported List of Countries: Malaysia (919-923)

China 920-925 MHz

The reader will operate in a sequential sequence over the frequencies specified in the channel table. The channel table is configurable in this region.

The dwell time per channel is less than 1800ms. The reader may dwell up to a maximum of 2000ms while performing Tag Access operations. The following table lists the available channels for this region.

Table 1.18: Supported Channel List

Channel	Frequency (MHz)
3	920.625
4	920.875
5	921.125
6	921.375
7	921.625
8	921.875
9	922.125
10	922.375
11	922.625
12	922.875
13	923.125
14	923.375
15	923.625
16	923.875
17	924.125
18	924.375

The Impinj RFID Reader will use the following default channel list for this region.

Table 1.19: Default Channel List

Channel	Frequency (MHz)
3	920.625
7	921.625
11	922.625
15	923.625

Supported List of Countries: China (920-925)

South Africa 915-919 MHz

The reader will operate in a random sequence over the frequencies specified in the channel table. The channel table is not configurable in this region.

The dwell time per channel is less than 200ms. The reader may dwell up to a maximum of 400ms while performing Tag Access operations. The following table lists the available channels for this region.

Table 1.20: Supported Channel List

Channel	Frequency (MHz)
1	915.6
2	915.8
3	916.0
4	916.2
5	916.4
6	916.6
7	916.8
8	917.0
9	917.2
10	917.4
11	917.6
12	917.8
13	918.0
14	918.2
15	918.4
16	918.6
17	918.8

The Impinj RFID Reader will use all supported channels by default.

Supported List of Countries: South Africa (915-919)

Brazil 902-907 and 915-928 MHz

The reader will operate in a random sequence over the frequencies specified in the channel table. The channel table is not configurable in this region.

The dwell time per channel is less than 200ms. The reader may dwell up to a maximum of 400ms while performing Tag Access operations. The following table lists the available channels for this region.

Table 1.21: Supported Channel List

Channel	Frequency (MHz)
1	902.75
2	903.25
3	903.75
4	904.25
5	904.75
6	905.25
7	905.75
8	906.25
9	906.75
10	907.25
26	915.25
27	915.75
28	916.25
29	916.75
30	917.25
31	917.75
Continued on next page	

Table 1.21 – continued from previous page

Channel	Frequency (MHz)
32	918.25
33	918.75
34	919.25
35	919.75
36	920.25
37	920.75
38	921.25
39	921.75
40	922.25
41	922.75
42	923.25
43	923.75
44	924.25
45	924.75
46	925.25
47	925.75
48	926.25
49	926.75
50	927.25

The Impinj RFID Reader will use all supported channels by default.

Supported List of Countries: Brazil (902-907), Brazil (915-928)

Thailand 920-925 MHz

The reader will operate in a random sequence over the frequencies specified in the channel table. The channel table is not configurable in this region.

The dwell time per channel is less than 200ms. The reader may dwell up to a maximum of 400ms while performing Tag Access operations. The following table lists the available channels for this region.

Table 1.22: Supported Channel List

Channel	Frequency (MHz)
1	920.25
2	920.75
3	921.25
4	921.75
5	922.25
6	922.75
7	923.25
8	923.75
9	924.25
10	924.75

The Impinj RFID Reader will use all supported channels by default.

Supported List of Countries: Thailand

Singapore 920-925 MHz

The reader will operate in a random sequence over the frequencies specified in the channel table. The channel table is not configurable in this region.

The dwell time per channel is less than 200ms. The reader may dwell up to a maximum of 400ms while performing Tag Access operations. The following table lists the available channels for this region.

Table 1.23: Supported Channel List

Channel	Frequency (MHz)
1	920.25
2	920.75
3	921.25
4	921.75
5	922.25
6	922.75
7	923.25
8	923.75
9	924.25
10	924.75

The Impinj RFID Reader will use all supported channels by default.

Supported List of Countries: Singapore (920-925)

Australia 920-926 MHz

The reader will operate in a random sequence over the frequencies specified in the channel table. The channel table is not configurable in this region.

The dwell time per channel is less than 200ms. The reader may dwell up to a maximum of 400ms while performing Tag Access operations. The following table lists the available channels for this region.

Table 1.24: Supported Channel List

Channel	Frequency (MHz)
1	920.25
2	920.75
3	921.25
4	921.75
5	922.25
6	922.75
7	923.25
8	923.75
9	924.25
10	924.75
11	925.25
12	925.75

The Impinj RFID Reader will use all supported channels by default.

Supported List of Countries: Australia (920-926)

India 865-867 MHz

The reader will operate in a sequential sequence over the frequencies specified in the channel table. The channel table is configurable in this region.

If only 1 channel is specified in the channel table, the reader switches off for 100ms after a dwell time of 4000ms on the channel.

The dwell time per channel is less than 3800ms. The reader may dwell up to a maximum of 4000ms while performing Tag Access operations. The following table lists the available channels for this region.

Table 1.25: Supported Channel List

Channel	Frequency (MHz)
1	865.1
4	865.7
7	866.3
10	866.9

The Impinj RFID Reader will use all supported channels by default.

Supported List of Countries: India

Uruguay 916-928 MHz

The reader will operate in a random sequence over the frequencies specified in the channel table. The channel table is not configurable in this region.

The dwell time per channel is less than 200ms. The reader may dwell up to a maximum of 400ms while performing Tag Access operations. The following table lists the available channels for this region.

Table 1.26: Supported Channel List

Channel	Frequency (MHz)
1	916.25
2	916.75
3	917.25
4	917.75
5	918.25
6	918.75
7	919.25
8	919.75
9	920.25
10	920.75
11	921.25
12	921.75
13	922.25
14	922.75
15	923.25
16	923.75
17	924.25
18	924.75
19	925.25
20	925.75
21	926.25
22	926.75
23	927.25

The Impinj RFID Reader will use all supported channels by default.

Supported List of Countries: Uruguay

Vietnam 920-925 MHz

The reader will operate in a random sequence over the frequencies specified in the channel table. The channel table is not configurable in this region.

The dwell time per channel is less than 200ms. The reader may dwell up to a maximum of 400ms while performing Tag Access operations. The following table lists the available channels for this region.

Table 1.27: Supported Channel List

Channel	Frequency (MHz)
1	920.25
2	920.75
3	921.25
4	921.75
5	922.25
6	922.75
7	923.25
8	923.75
9	924.25
10	924.75

The Impinj RFID Reader will use all supported channels by default.

Supported List of Countries: Vietnam (920-925)

Israel 915-917 MHz

The reader will operate in a sequential sequence over the frequencies specified in the channel table. The channel table is configurable in this region.

The dwell time per channel is less than 0ms. The reader may dwell up to a maximum of 0ms while performing Tag Access operations. The following table lists the available channels for this region.

Table 1.28: Supported Channel List

Channel	Frequency (MHz)
1	916.25

The Impinj RFID Reader will use all supported channels by default.

Supported List of Countries: Israel

Philippines 918-920 MHz

The reader will operate in a sequential sequence over the frequencies specified in the channel table. The channel table is configurable in this region.

The dwell time per channel is less than 3800ms. The reader may dwell up to a maximum of 4000ms while performing Tag Access operations. The following table lists the available channels for this region.

Table 1.29: Supported Channel List

Channel	Frequency (MHz)
1	918.25
2	918.75
3	919.25
4	919.75

The Impinj RFID Reader will use all supported channels by default.

Supported List of Countries: Philippines

Indonesia 923-925 MHz

The reader will operate in a random sequence over the frequencies specified in the channel table. The channel table is not configurable in this region.

The dwell time per channel is less than 200ms. The reader may dwell up to a maximum of 400ms while performing Tag Access operations. The following table lists the available channels for this region.

Table 1.30: Supported Channel List

Channel	Frequency (MHz)
1	923.25
2	923.75
3	924.25
4	924.75

The Impinj RFID Reader will use all supported channels by default.

Supported List of Countries: Indonesia

New Zealand 921.5-928 MHz

The reader will operate in a random sequence over the frequencies specified in the channel table. The channel table is not configurable in this region.

The dwell time per channel is less than 200ms. The reader may dwell up to a maximum of 400ms while performing Tag Access operations. The following table lists the available channels for this region.

Table 1.31: Supported Channel List

Channel	Frequency (MHz)
1	922.25
2	922.75
3	923.25
4	923.75
5	924.25
6	924.75
7	925.25
8	925.75
9	926.25
10	926.75
11	927.25

The Impinj RFID Reader will use all supported channels by default.

Supported List of Countries: New Zealand (921-928)

Japan 916-921 MHz, no LBT

The reader will operate in a sequential sequence over the frequencies specified in the channel table. The channel table is configurable in this region.

If only 1 channel is specified in the channel table, the reader switches off for 50ms after a dwell time of 4000ms on the channel.

The dwell time per channel is less than 3800ms. The reader may dwell up to a maximum of 4000ms while performing Tag Access operations. The following table lists the available channels for this region.

Table 1.32: Supported Channel List

Channel	Frequency (MHz)
6	916.8
12	918.0
18	919.2
24	920.4

The Impinj RFID Reader will use all supported channels by default.

Supported List of Countries: Japan (916-921)

Peru 916-928 MHz

The reader will operate in a random sequence over the frequencies specified in the channel table. The channel table is not configurable in this region.

The dwell time per channel is less than 200ms. The reader may dwell up to a maximum of 400ms while performing Tag Access operations. The following table lists the available channels for this region.

Table 1.33: Supported Channel List

Channel	Frequency (MHz)
1	916.25
2	916.75
3	917.25
4	917.75
5	918.25
6	918.75
7	919.25
8	919.75
9	920.25
10	920.75
11	921.25
12	921.75
13	922.25
14	922.75
15	923.25
16	923.75
17	924.25
18	924.75
19	925.25
20	925.75
21	926.25
22	926.75
23	927.25

The Impinj RFID Reader will use all supported channels by default.

Supported List of Countries: Peru

Russia 916-921 MHz

The reader will operate in a sequential sequence over the frequencies specified in the channel table. The channel table is configurable in this region.

If only 1 channel is specified in the channel table, the reader switches off for 100ms after a dwell time of 4000ms on the channel.

The dwell time per channel is less than 3800ms. The reader may dwell up to a maximum of 4000ms while performing Tag Access operations. The following table lists the available channels for this region.

Table 1.34: Supported Channel List

Channel	Frequency (MHz)
1	916.2
2	917.4
3	918.6
4	919.8

The Impinj RFID Reader will use all supported channels by default.

Supported List of Countries: Russian Federation (916-921)

Custom

See the *Configure the Indy SiP with a Custom Region* feature for a detailed usage description.

1.5.13 Error Codes

This section describes error codes returned from API function calls.

E_IPJ_ERROR_SUCCESS

Success

- Error: 0x00000000
- Category: 0
- Diagnostics: NA
- Description: Success

E_IPJ_ERROR_GENERAL_ERROR

General Error

- Error: 0x00000001
- Category: 0
- Diagnostics: NA
- Description: General Error

E_IPJ_ERROR_SET_KEY_INVALID

Set Key Invalid

- Error: 0x00000002
- Category: 0
- Diagnostics: NA
- Description: Set Key Invalid

E_IPJ_ERROR_SET_KEY_READ_ONLY

Set Key Read Only

- Error: 0x00000003
- Category: 0
- Diagnostics: NA
- Description: Set Key Read Only

E_IPJ_ERROR_SET_KEY_OUT_OF_RANGE

Set Key Out Of Range

- Error: 0x00000004
- Category: 0
- Diagnostics: Key, Min Value, Max Value, User Value
- Description: Set Key Out Of Range

E_IPJ_ERROR_GET_KEY_INVALID

Get Key Invalid

- Error: 0x00000005
- Category: 0
- Diagnostics: NA
- Description: Get Key Invalid

E_IPJ_ERROR_GET_KEY_WRITE_ONLY

Get Key Write Only

- Error: 0x00000006
- Category: 0
- Diagnostics: NA
- Description: Get Key Write Only

E_IPJ_ERROR_COMMAND_INVALID

Command Invalid

- Error: 0x00000007
- Category: 0
- Diagnostics: NA
- Description: Command Invalid

E_IPJ_ERROR_COMMAND_START_FAILURE

Command Start Failure

- Error: 0x00000008
- Category: 0
- Diagnostics: NA
- Description: Command Start Failure

E_IPJ_ERROR_COMMAND_DECODE_FAILURE

Command Decode Failure

- Error: 0x00000009
- Category: 0
- Diagnostics: NA
- Description: Command Decode Failure

E_IPJ_ERROR_COMMAND_ENCODE_FAILURE

Command Decode Failure

- Error: 0x0000000A
- Category: 0
- Diagnostics: NA
- Description: Command Decode Failure

E_IPJ_ERROR_COMMAND_STALLED

Command Stalled

- Error: 0x0000000B
- Category: 0
- Diagnostics: NA
- Description: Command Stalled

E_IPJ_ERROR_VALUE_INVALID

Invalid Value

- Error: 0x0000000C
- Category: 0
- Diagnostics: NA
- Description: Invalid Value

E_IPJ_ERROR_MORE_THAN_ONE_COMMAND_RECEIVED

More than one command received

- Error: 0x0000000D
- Category: 0
- Diagnostics: NA
- Description: More than one command received

E_IPJ_ERROR_NOT_IMPLEMENTED

The feature requested is not available in this release

- Error: 0x0000000E
- Category: 0
- Diagnostics: NA
- Description: The feature requested is not available in this release

E_IPJ_ERROR_INVALID_PRODUCT_CONFIGURATION

The product contains an invalid configuration

- Error: 0x0000000F
- Category: 0
- Diagnostics: NA
- Description: The product contains an invalid configuration

E_IPJ_ERROR_INVALID_FACTORY_SETTINGS

The product contains invalid factory settings

- Error: 0x00000010
- Category: 0
- Diagnostics: NA
- Description: The product contains invalid factory settings

E_IPJ_ERROR_RESPONSE_ENCODE_FAILURE

The device encountered an error while trying to encode a Report/Response

- Error: 0x00000011
- Category: 0
- Diagnostics: NA
- Description: The device encountered an error while trying to encode a Report/Response

E_IPJ_ERROR_COMMAND_VERIFY_FAILURE

The device detected that the intended data was not properly written

- Error: 0x00000012
- Category: 0
- Diagnostics: NA
- Description: The device detected that the intended data was not properly written

E_IPJ_ERROR_INTERNAL_NON_RECOVERABLE

The device encountered an internal non-recoverable error.

- Error: 0x00000013
- Category: 0
- Diagnostics: NA
- Description: The device encountered an internal non-recoverable error.

E_IPJ_ERROR_TEMPLATE_DECODE_FAILURE

The device was unable to properly decode the command template

- Error: 0x00000014
- Category: 0
- Diagnostics: NA
- Description: The device was unable to properly decode the command template

E_IPJ_ERROR_SYSTEM_IN_ERROR_STATE

A non-recoverable error has occurred and the system is in an error state. See `E_IPJ_KEY_SYSTEM_ERROR`. Issue a `E_IPJ_ACTION_CLEAR_ERROR` to clear the error.

- Error: 0x00000015
- Category: 0
- Diagnostics: NA
- Description: A non-recoverable error has occurred and the system is in an error state. See `E_IPJ_KEY_SYSTEM_ERROR`. Issue a `E_IPJ_ACTION_CLEAR_ERROR` to clear the error.

E_IPJ_ERROR_TEST_ERROR

A general test error has occurred.

- Error: 0x00000016
- Category: 0
- Diagnostics: NA
- Description: A general test error has occurred.

E_IPJ_ERROR_STORED_SETTING_DECODE

An error occurred while decoding a stored setting. The offending setting Key code is reported in the first error parameter.

- Error: 0x00000017
- Category: 0
- Diagnostics: NA
- Description: An error occurred while decoding a stored setting. The offending setting Key code is reported in the first error parameter.

E_IPJ_ERROR_VALUE_INDEX_OUT_OF_RANGE

A Set or Get value_index parameter is out of range

- Error: 0x00000018
- Category: 0
- Diagnostics: NA
- Description: A Set or Get value_index parameter is out of range

E_IPJ_ERROR_BANK_INDEX_OUT_OF_RANGE

A Set or Get bank_index parameter is out of range

- Error: 0x00000019
- Category: 0
- Diagnostics: NA
- Description: A Set or Get bank_index parameter is out of range

E_IPJ_ERROR_INVALID_PRODUCT_CALIBRATION

The product contains invalid calibration data

- Error: 0x0000001A
- Category: 0
- Diagnostics: NA
- Description: The product contains invalid calibration data

E_IPJ_ERROR_REPORT_SIZE_WOULD_OVERFLOW

The requested report fields and data would overflow the buffer.

- Error: 0x0000001B
- Category: 0
- Diagnostics: NA
- Description: The requested report fields and data would overflow the buffer.

E_IPJ_ERROR_GEN2_TAG_OTHER_ERROR

Catch-all for tag errors not covered by other codes

- Error: 0x01000001
- Category: 1
- Diagnostics: NA
- Description: Catch-all for tag errors not covered by other codes

E_IPJ_ERROR_GEN2_TAG_MEMORY_OVERRUN

The specified memory location does not exist or the EPC length field is not supported by the Tag

- Error: 0x01000002
- Category: 1
- Diagnostics: NA
- Description: The specified memory location does not exist or the EPC length field is not supported by the Tag

E_IPJ_ERROR_GEN2_TAG_MEMORY_LOCKED

The specified memory location is locked and/or permalocked and is either not writeable or not readable

- Error: 0x01000003
- Category: 1
- Diagnostics: NA
- Description: The specified memory location is locked and/or permalocked and is either not writeable or not readable

E_IPJ_ERROR_GEN2_TAG_INSUFFICIENT_POWER

The Tag has insufficient power to perform the memory-write operation

- Error: 0x01000004
- Category: 1
- Diagnostics: NA
- Description: The Tag has insufficient power to perform the memory-write operation

E_IPJ_ERROR_GEN2_TAG_NON_SPECIFIC_ERROR

The Tag does not support error-specific codes

- Error: 0x01000005
- Category: 1
- Diagnostics: NA
- Description: The Tag does not support error-specific codes

E_IPJ_ERROR_API_DEVICE_NOT_INITIALIZED

Device is not initialized

- Error: 0x02000001
- Category: 2
- Diagnostics: NA
- Description: Device is not initialized

E_IPJ_ERROR_API_SERIAL_PORT_ERROR

Serial Port Error

- Error: 0x02000002
- Category: 2
- Diagnostics: NA
- Description: Serial Port Error

E_IPJ_ERROR_API_CONNECTION_READ_TIMEOUT

Connection Read Timeout

- Error: 0x02000003
- Category: 2
- Diagnostics: NA
- Description: Connection Read Timeout

E_IPJ_ERROR_API_CONNECTION_WRITE_TIMEOUT

Connection Write Timeout

- Error: 0x02000004
- Category: 2
- Diagnostics: NA
- Description: Connection Write Timeout

E_IPJ_ERROR_API_CONNECTION_WRITE_ERROR

Connection Write Error

- Error: 0x02000005
- Category: 2
- Diagnostics: NA
- Description: Connection Write Error

E_IPJ_ERROR_API_RX_BUFF_TOO_SMALL

Receive buffer too small

- Error: 0x02000006
- Category: 2
- Diagnostics: NA
- Description: Receive buffer too small

E_IPJ_ERROR_API_MESSAGE_INVALID

Invalid Message

- Error: 0x02000007
- Category: 2
- Diagnostics: NA
- Description: Invalid Message

E_IPJ_ERROR_API_NO_HANDLER

No user handler has been registered to handle this action

- Error: 0x02000008
- Category: 2
- Diagnostics: NA
- Description: No user handler has been registered to handle this action

E_IPJ_ERROR_API_INVALID_LOADER_BLOCK

The API was passed an invalid loader block

- Error: 0x02000009
- Category: 2
- Diagnostics: NA
- Description: The API was passed an invalid loader block

E_IPJ_ERROR_API_RESPONSE_MISMATCH

The response received does not match the command that was sent

- Error: 0x0200000A
- Category: 2
- Diagnostics: NA
- Description: The response received does not match the command that was sent

E_IPJ_ERROR_API_INVALID_PARAMETER

The API called detected an invalid parameter (a NULL pointer, an array length that is too large, etc.)

- Error: 0x0200000B
- Category: 2
- Diagnostics: NA
- Description: The API called detected an invalid parameter (a NULL pointer, an array length that is too large, etc.)

E_IPJ_ERROR_API_NON_LT_PACKET_DETECTED

The receive routine detected a non-LT packet

- Error: 0x0200000C
- Category: 2
- Diagnostics: NA
- Description: The receive routine detected a non-LT packet

E_IPJ_ERROR_IRI_FRAME_DROPPED

Missing/Malformed IRI Packet

- Error: 0x03000001
- Category: 3
- Diagnostics: NA
- Description: Missing/Malformed IRI Packet

E_IPJ_ERROR_IRI_FRAME_INVALID

The calculated IRI frame CRC or parity did not match the value that was sent

- Error: 0x03000002
- Category: 3
- Diagnostics: NA
- Description: The calculated IRI frame CRC or parity did not match the value that was sent

E_IPJ_ERROR_MAC_GENERAL

General Catch all for all MAC Errors

- Error: 0x04000001
- Category: 4
- Diagnostics: NA
- Description: General Catch all for all MAC Errors

E_IPJ_ERROR_MAC_CRC_MISMATCH

CRC Mismatch on Tag Response

- Error: 0x04000002
- Category: 4
- Diagnostics: NA
- Description: CRC Mismatch on Tag Response

E_IPJ_ERROR_MAC_NO_TAG_RESPONSE

No Tag Response

- Error: 0x04000003
- Category: 4
- Diagnostics: NA
- Description: No Tag Response

E_IPJ_ERROR_MAC_TAG_LOST

Tag Lost

- Error: 0x04000004
- Category: 4
- Diagnostics: NA
- Description: Tag Lost

E_IPJ_ERROR_BTS_DEVICE_WATCHDOG_RESET

Device has experienced a watchdog reset due to a hard lock

- Error: 0x05000001
- Category: 5
- Diagnostics: NA
- Description: Device has experienced a watchdog reset due to a hard lock

E_IPJ_ERROR_BTS_VALUE_INVALID

Command contains an invalid value

- Error: 0x05000002
- Category: 5
- Diagnostics: NA
- Description: Command contains an invalid value

E_IPJ_ERROR_BTS_FLASH_WRITE

Error writing to flash

- Error: 0x05000003
- Category: 5
- Diagnostics: NA
- Description: Error writing to flash

E_IPJ_ERROR_BTS_FLASH_READ

Error reading from flash

- Error: 0x05000004
- Category: 5
- Diagnostics: NA
- Description: Error reading from flash

E_IPJ_ERROR_BTS_FLASH_ADDRESS

Address is protected

- Error: 0x05000005
- Category: 5
- Diagnostics: NA
- Description: Address is protected

E_IPJ_ERROR_BTS_FLASH_ERASE

Error erasing flash page

- Error: 0x05000006
- Category: 5
- Diagnostics: NA
- Description: Error erasing flash page

E_IPJ_ERROR_BTS_UNKNOWN_COMMAND

Device received an unknown command

- Error: 0x05000007
- Category: 5
- Diagnostics: NA
- Description: Device received an unknown command

E_IPJ_ERROR_BTS_COMMAND_DECODE_FAILURE

Device was unable to decode the command

- Error: 0x05000008
- Category: 5
- Diagnostics: NA
- Description: Device was unable to decode the command

E_IPJ_ERROR_TRANSCEIVER_FAILURE

Error communicating with device transceiver

- Error: 0x06000001
- Category: 6
- Diagnostics: NA
- Description: Error communicating with device transceiver

E_IPJ_ERROR_LIMIT_PA_TEMPERATURE_MAX

Device PA temperature exceeded part specification

- Error: 0x07000001
- Category: 7
- Diagnostics: NA
- Description: Device PA temperature exceeded part specification

1.5.14 Platform and Report Handlers

Platform Handler Definitions

READER_CONTEXT

typedef void * **IPJ_READER_CONTEXT**

IPJ_READER_CONTEXT is a HANDLE to the serial port.

READER_IDENTIFIER

typedef void * **IPJ_READER_IDENTIFIER**

IPJ_READER_IDENTIFIER is a reference used by the platform to open serial port
IPJ_READER_IDENTIFIER is also used by report handler to allow the application to associate reports with a given reader in systems with multiple readers.

Platform Handler Interface

PLATFORM_OPEN_PORT_HANDLER

```
typedef uint32_t (* PLATFORM_OPEN_PORT_HANDLER)(IPJ_READER_CONTEXT *reader_context,
IPJ_READER_IDENTIFIER reader_identifer, ipj_connection_type connection_type, ipj_connection_params
*params)
```

PLATFORM_CLOSE_PORT_HANDLER

```
typedef uint32_t (* PLATFORM_CLOSE_PORT_HANDLER)(IPJ_READER_CONTEXT reader_context)
```

PLATFORM_TRANSMIT_HANDLER

```
typedef uint32_t (* PLATFORM_TRANSMIT_HANDLER)(IPJ_READER_CONTEXT reader_context, uint8_t
*message_buffer, uint16_t buffer_size, uint16_t *number_bytes_written)
```

PLATFORM_RECEIVE_HANDLER

```
typedef uint32_t (* PLATFORM_RECEIVE_HANDLER)(IPJ_READER_CONTEXT reader_context, uint8_t
*message_buffer, uint16_t buffer_size, uint16_t *number_bytes_received, uint16_t timeout_ms)
```

PLATFORM_TIMESTAMP_MS_HANDLER

```
typedef uint32_t (* PLATFORM_TIMESTAMP_MS_HANDLER)()
```

PLATFORM_SLEEP_MS_HANDLER

```
typedef void(* PLATFORM_SLEEP_MS_HANDLER)(uint32_t milliseconds)
```

Report Handler Interface

REPORT_HANDLER

```
typedef ipj_error (* REPORT_HANDLER)(struct ipj_iri_device *iri_device, ipj_report_id report_id, void *report)
```

DIAGNOSTIC_HANDLER

```
typedef void(* DIAGNOSTIC_HANDLER)(struct ipj_iri_device *iri_device, ipj_error error)
```

1.5.15 Stored Settings

Stored settings provides a way for the user to save default key settings to the device. On boot up these settings will automatically be loaded in to the specified keys, thereby eliminating the need for the Host to issue any type of *set* for those keys.

File Format

The settings XML file has a *Settings* root with any number of *setting* sub-elements.

```
<?xml version="1.0" ?>
<Settings>
  <setting>
    <key>E_IPJ_KEY_1</key>
    <bank>0</bank>
    <values>0</values>
  </setting>
  <setting>
    <key>E_IPJ_KEY_2</key>
    <bank>0</bank>
    <values>0</values>
  </setting>
  ...
  <setting>
    <key>E_IPJ_KEY_N</key>
    <bank>0</bank>
    <values>0</values>
  </setting>
</Settings>
```

General Guidelines

- Any *Key* that has **Read/Write** Permissions *can* be stored
- Conversely, **Read Only** keys *cannot* be stored and should not be included in a settings XML file
- If the Application detects a bad setting then *E_IPJ_KEY_LAST_ERROR* will be set and the setting will remain at the default value.
- *E_IPJ_KEY_LAST_ERROR* *value_index* description:
 - *value_index[0]* will be set to *E_IPJ_ERROR_STORED_SETTING_DECODE*
 - *value_index[1]* will be set to the offending *Key* code

Key Requirements

- *key* is a required tag
- *key* is the actual IRI *Key* name, *not* the numerical **Key Id**

Bank Requirements

- *bank* is a required tag for *Keys* that have a **Bank Count** greater than 0
- Conversely, the *bank* tag is ignored and should be omitted for keys with a **Bank Count** of 0
- If **Bank Count** is greater than 0 and *bank* is not present then a default value of 0 will be used

Values Requirements

- *values* is a required tag
- *values* is a single entry for *Keys* that have a **Value Count** of 1
- *values* is a comma separated list for *Keys* that have a **Value Count** greater than 1
- *values* can support IRI *Defines* if appropriate for the given key
- *values* can support hexadecimal entries that have the '0x' prefix
- If **Value Count** is greater than 1, and not all values are provided in the comma separated list, then all remaining values will automatically be zero-filled
- Invalid entries for *values* will result in undefined behavior

Examples

Simple Auto Start

This example will:

- Demonstrate keys with **Value Counts** of 1
- Demonstrate keys with **Bank Counts** of 0 and 1
- Set the region to China
- Set the antenna TX power to 21.00 dBm
- Configure inventory to automatically start when the device boots up

Note: *E_IPJ_KEY_ANTENNA_TX_POWER* has a **Bank Count** of 1 so the *bank* tag is used. The other keys have a **Bank Count** of 0 so the *bank* tag is omitted.

```
<?xml version="1.0" ?>
<Settings>
  <setting>
    <key>E_IPJ_KEY_REGION_ID</key>
    <values>E_IPJ_REGION_CHINA_920_925_MHZ</values>
  </setting>
  <setting>
```

```

    <key>E_IPJ_KEY_ANTENNA_TX_POWER</key>
    <bank>0</bank>
    <values>2100</values>
  </setting>
  <setting>
    <key>E_IPJ_KEY_ONBOOT_START_ACTION</key>
    <values>E_IPJ_ACTION_INVENTORY</values>
  </setting>
</Settings>

```

Channel Table and GPI Triggering

This example will:

- Demonstrate a key with **Value Count** greater than 1
- Demonstrate keys with **Bank Counts** greater than 1
- Demonstrate a comma separated list
- Set the region to Japan
- Set up a custom channel table
- Configure a GPI trigger for starting and stopping inventory

```

<?xml version="1.0" ?>
<Settings>

  <!-- Set up a 3 channel table for Japan -->
  <setting>
    <key>E_IPJ_KEY_REGION_ID</key>
    <values>E_IPJ_REGION_JAPAN_916_921_MHZ_NO_LBT</values>
  </setting>
  <setting>
    <key>E_IPJ_KEY_REGION_CHANNEL_TABLE</key>
    <values>6, 18, 12</values>
  </setting>
  <setting>
    <key>E_IPJ_KEY_REGION_CHANNEL_TABLE_SIZE</key>
    <values>3</values>
  </setting>

  <!-- Set up a GPI Inventory Start on Port 4 (bank 4) -->
  <setting>
    <key>E_IPJ_KEY_GPIO_MODE</key>
    <bank>4</bank>
    <values>E_IPJ_GPIO_MODE_INPUT_ACTION</values>
  </setting>
  <setting>
    <key>E_IPJ_KEY_GPIO_STATE</key>
    <bank>4</bank>
    <values>E_IPJ_GPIO_STATE_LO</values>
  </setting>
  <setting>
    <key>E_IPJ_KEY_GPIO_HI_ACTION</key>
    <bank>4</bank>
    <values>E_IPJ_GPI_ACTION_START_INVENTORY</values>
  </setting>

```

```

<setting>
  <key>E_IPJ_KEY_GPIO_DEBOUNCE_MS</key>
  <bank>4</bank>
  <values>10</values>
</setting>

<!-- Set up a GPI Inventory Stop on Port 1 (bank 1) -->
<setting>
  <key>E_IPJ_KEY_GPIO_MODE</key>
  <bank>1</bank>
  <values>E_IPJ_GPIO_MODE_INPUT_ACTION</values>
</setting>
<setting>
  <key>E_IPJ_KEY_GPIO_STATE</key>
  <bank>1</bank>
  <values>E_IPJ_GPIO_STATE_LO</values>
</setting>
<setting>
  <key>E_IPJ_KEY_GPIO_HI_ACTION</key>
  <bank>1</bank>
  <values>E_IPJ_GPI_ACTION_STOP_INVENTORY</values>
</setting>
</Settings>

```

Note: In this example GPI triggering will not start until a *start* command is issued with an *E_IPJ_ACTION_GPIO* argument

Creating, Loading and Saving

A stored settings XML file must be converted to a loader image before it can be used. Stored settings loader images are like any other application image and can be loaded as such.

The Indy Demo Tool’s “Image Loader” tab has controls for stored settings.

- A button for saving a stored settings XML file from the attached device, with an optional checkbox to save the loader image as well
- A button for making a loader image binary from a supplied stored settings XML file

Note: To create a stored settings image the Indy Demo Tool must be connected to the IRI device

Note: All *values* in a settings file that has been saved via the Indy Demo Tool will be in **decimal** format

Empty Settings

If the Indy SiP does not have any stored settings then using the Indy Demo Tool save settings button will result in a file as follows.

```

<?xml version="1.0" ?>
<Settings/>

```

- A new Indy SiP will be in this state.
- Loading an empty settings is one way to restore factory defaults, as an alternate to the `E_IPJ_RESET_TYPE_FACTORY_RESTORE` argument to the `reset` command.

1.5.16 Memory Usage

The following table lists verified ITK-C host platforms and corresponding library sizes.

Notes:

- These results come from library version 1.1.2.240
- **All `config.h` compile time switches were disabled**
 - The `ENABLE_FW_UPDATES` compile time switch increases the library sizes reported below by approximately 750 bytes
- The ITK-C library is comprised of multiple files
- The ITK-LT-C library has been reduced to one file
- Linux, Windows, and OSX ports are available now
- Embedded MCU ports will be included in a future version of the ITK

If you have questions please submit a support ticket at support.impinj.com.

Table 1.35: Library Sizes

Library	Build Machine	Target Platform	Target Architecture	Target Processor	Toolchain	Optimizations	Library Contents	Size (Bytes)
ITK-C	Windows 7 + Cygwin	Windows	32-bit	x86	GCC 4.5.3	For size (-Os)	iri.o, pb_decode.o, pb_encode.o, messages.pb.o, commands.pb.o, packet.pb.o	11,868
ITK-C	CentOS 6.5	Linux	32-bit	x86	GCC 4.4.7	For size (-Os)	iri.o, pb_decode.o, pb_encode.o, messages.pb.o, commands.pb.o, packet.pb.o	11,396
ITK-C	Apple Mac 10.9.2	OSX	32-bit	x86	GCC (Apple LLVM 5.1 - Clang 503.0.40)	For size (-Os)	iri.o, pb_decode.o, pb_encode.o, messages.pb.o, commands.pb.o, packet.pb.o	14,817
ITK-C	Ubuntu 12.4.3	STM32F0 DISCOVERY Board	32-bit	ARM Cortex-M0 (STM32F051)	GCC ARM Embedded 4.8.3	For size (-Os)	iri.o, pb_decode.o, pb_encode.o, messages.pb.o, commands.pb.o, packet.pb.o	9,333
ITK-C	Windows 7 + Atmel-Studio 6.1	AVR XMEGA-A1 Xplained	8-bit	Atmel AVR (ATxmega1284L)	AVR/GNU C Compiler	For size (-Os)	iri.o, pb_decode.o, pb_encode.o, messages.pb.o, commands.pb.o, packet.pb.o	15,178

Notices:

Copyright © 2013-2015, Impinj, Inc. All rights reserved.

The information contained in this document is confidential and proprietary to Impinj, Inc. This document is conditionally issued, and neither receipt nor possession hereof confers or transfers any right in, or license to, use the subject matter of any drawings, design, or technical information contained herein, nor any right to reproduce or disclose any part of the contents hereof, without the prior written consent of Impinj and the authorized recipient hereof.

Impinj reserves the right to change its products and services at any time without notice.

Impinj assumes no responsibility for customer product design or for infringement of patents and/or the rights of third parties, which may result from assistance provided by Impinj. No representation of warranty is given and no liability is assumed by Impinj with respect to accuracy or use of such information.

These products are not designed for use in life support appliances, devices, or systems where malfunction can reasonably be expected to result in personal injury.

This product is covered by one or more of the following U.S. patents. Other patents pending.
'<<http://www.impinj.com/patents>>' _