JSR Common API Library
Software Development Kit

Programmers' Reference Manual

SDK Version 1.3.0.0
March, 2011

www.JSRUltrasonics.com

TechSupport@JSRUltrasonics.com

Table of Contents

1 Introduction

1.1 Important Notice: See Chapter 3 about 32 bit vs 64 bit applications and OS

1.2 Scope and Contents of this Document

This document describes the JSR Common API Library and related Software Development Kit. These items are software products of the JSR Ultrasonics product line of Imaginant Inc. of Pittsford, New York, USA (herein called JSR), www.Imaginant.com

This document is intended as a reference for persons designing and writing a Windows Application that interfaces with the JSR Common API Library in order to control one or more JSR Pulser-Receiver instruments, such as the PRC50, DPR300, or DPR500.

Important additional details of using the JSR Common API Library are in the separate document: JSR_API_PropertiesReferenceManual.pdf

1.3 JSR Common API Library Overview

The JSR Common API Library was designed to give 3rd party developers an easy way to interface their software to JSR Instruments through standard Windows DLL's (JSR_Commonxxyy.dll). This system gives the following advantages:

- Simplify integration of JSR products into a customer's host application
- Hide details of JSR product hardware interface mechanism (PCI bus, RS-232, etc.) and protocol and sequencing that are not relevant to an application
- Provide forward compatibility for future products via a standardized interface
- Reduce development time by providing a single interface design for multiple models of JSR products
- Allow intuitive control of more than one JSR instrument at the same time, even if different models
- Provide example code that can be used by external developers to test a system
- Provide example code that can be modified by external developers to their needs

The JSR_Simple Library provides a simplified method of accessing the JSR_Common library interface.

- Removes the overhead of maintaining object handle information by the application layer.
- Property methods are performed on a per channel basis.
- Property data types are automatically determined by the method prototypes.

The JSR_Simple_ActiveX Library interface provides access to the JSR_Simple interface by ActiveX friendly development environemts such as Visual Basic.

1.4 Software Licensing

Source code of the JSR Common API SDK (herein the SDK) remains the Copyright of Imaginant Inc. and is not part of the released or licensed components of the SDK. The Sample Code may be used and modified by application developers.

License to use the SDK and distribute components of it are granted to the followingparties:

Any party that has purchased one or more JSR products.

Any party that has purchased or leased a system that includes one or more JSR products.

Any party that is considering purchase of a JSR product.

Any party writing an application that will connect to JSR products.

External application developers are hereby granted a license to use the SDK within their software product and distribute any and all parts of the SDK with their product. A small number of third-party dynamic libraries may need to be used with the SDK. External developers will need to acquire those libraries directly from that source, for example a Microsoft® CD-ROM or the Microsoft® web site.

1.5 SDK Distribution and Support

The JSR Common API Library SDK is freely available for developers or individuals testing or using, or planning to use Imaginant's JSR Ultrasonics products. Contact TechSupport@Imaginant.com

- The CDROM that is shipped with purchased JSR ultrasonics products.
- Free download from www.Imaginant.com as a zip file.
- A zip file as an email attachment when requested.
- A CDROM can be shipped when requested.

1.6 Support

The most recent versions of these documents will be available via download from www.Imaginant.com
JSR_API_ProgrammersReferenceManual.pdf
JSR_API_PropertiesReferenceManual.pdf
JSR_API_FAQ.pdf

Specific questions about implementation may be directed by email to TechSupport@Imaginant.com

1.7 Contents

The following files and folders are a complete list of the contents of the SDK.

JSR_API_ProgrammersReferenceManual.pdf (This document)
JSR_API_PropertiesReferenceManual.pdf
JSR_CommonXXYY.dll The dynamic library that implements the code.
JSR_Common.lib: For Static Linking, host application should link with this file.
JSR_Loader.c source code to provide run-time loading of JSR_CommonXXYY.dll
JSR_API_FAQ.pdf Frequently Asked Questions
ReleaseNotes.txt defining new features and known bugs.
Example source code, with project files.

Header files to be linked into the customer's code:

JSR_Types.h	defines data types
JSR_Status.h	defines error code numbers
JSR_PropertyID.h	defines constants that identify specific properties
Either:	
JSR_Loader.h	header file for JSR_Loader.c
OR	
JSR_Common.h	defines interface functions

1.8 Supported Platform and Operating System Environments

1.8.1 Hardware Supported

A Pentium® or compatible, or higher performance computer.
Minimum 256 MB RAM
Minimum 500 MB Disk space
For PRC50: PCI Bus with at least one full slot available
For DPR300 and DPR500: At least one RS-232 serial port or a USB Serial Port dongle

1.8.2 Operating Systems Supported

Windows XP Professional and Home (32 and 64 bits)
Windows Embedded (32 and 64 bits)
Windows Vista Professional and Home (32 and 64 bits)
Windows 7 Professional and Home (32 and 64 bits)

2 Overview

2.1 Important Notice: See Chapter 3 about 32 bit vs 64 bit applications and OS

2.2 Software Development Environment

The JSR_CommonXXYY.dll (the implementation of the library) was designed and built under Microsoft Visual C++ 6.0, and the SDK as a whole is packaged for easy integration into other Visual C++ 6.0 projects. However, the DLL interface uses ANSI standard C, and can be interfaced by any language or development environment that can connect to a C interface packaged in a DLL.

The source code of the sample code can be used with any development environment using the "C" language. However, sample code project files are included only for Microsoft® Developer Studio. For developers using non C/C++ languages, SWIG (http://www.swig.org) is a good resource for interfacing C/C++ systems to programs developed in other languages.

2.3 Using the JSR Common API Library

The JSR Common API Library was designed to be easy to use and work with for application developers. To this end, the entire library is designed as a set of objects, each with a set of properties. Each type of object has a list of allowed property ID's, which are specified in JSR_PropertyID.h.

Through this system the instruments are controlled as state machines, whose states change based on property settings. Each object has its own properties:

Read-only properties report such items as capabilities and serial numbers.

Read-write properties that are used by an application to control settings in the instrument.

2.4 Loading the JSR Common API Library

There are two ways for your application to connect with the JSR Common API Library, using either static link-time binding where your code gets linked with JSR_CommonXXYY.lib or dynamic run-time loading where your application loads pointers to the methods within the JSR_CommonXXYY.dll at launch-time using sample code provided in the SDK. Details are further below.

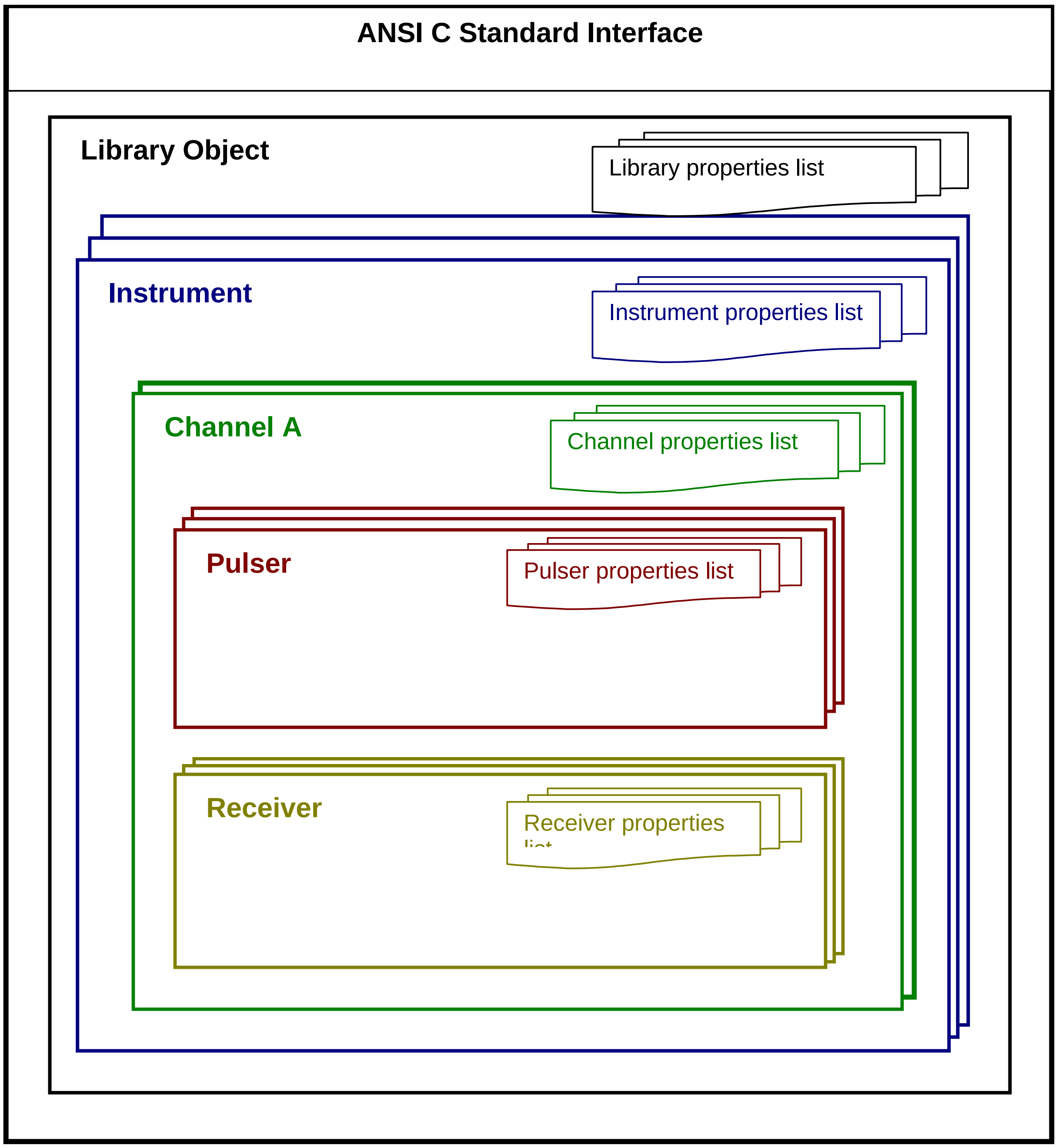
2.5 Structure of Objects in JSR Common API Library

There is a hierarchy of objects containing objects in the JSR Common API, as follows:

Library Object	The library object can contain 0 to several Instrument objects. These Instruments can be any combination of Instruments supported by the version of the Library.
Instrument Object	An instrument object can contain 1 or 2 channel objects, depending on the instrument model. PRC50's and DPR300's can have only one channel. DPR500's can be configured to have one or two channels.
Channel Object	A channel object can contain 1 to 2 Receivers, and 1 to 16 Pulsers. The PRC50 and DPR300 products can have only one channel, which has only one pulser and one receiver. A DPR500 can have one or two channels and a channel may have several pulsers if connected to a JSR MUX1600 product. Remember that either channel of a DPR500 instrument may also be configured **without** a pulser or without a receiver.

Pulser Object	This object manages generation of pulses, and can not have child objects.
Receiver Object	This object manages ultrasonic pulse receivers, and can not have child objects.

2.6 Diagram of the Structure of Objects in JSR Common API Library



2.7 Introduction to general function calls

The host application opens and closes the JSR Common API Library with two functions:

```
JSR_OpenLibrary(…)
JSR_CloseLibrary(…)
```

NOTE: Closing the Library purposely leaves the instruments in whatever state was last commanded.

The host application must read a list of handle(s) to instrument objects:

```
JSR_GetInt32(JSR_LIBRARY_HANDLE, JSR_ID_LibraryInstrumentHandles,
handleCount, &appsInstrumentHandleArray)
```

The host application must read a list of handle(s) to channel, receiver, and pulser objects from each object's parent object by calling:

```
JSR_GetInt32(instrumentHandle, JSR_ID_OfSpecifiedHandleList,
handleCount, &appsHandleArray)
```

The host application must open and close instrument, channel, receiver, and pulser objects using an object handle acquired as above:

```
JSR_OpenObject(myObjectHandle, ObjectOpenOptions)
JSR_CloseObject (myObjectHandle)
```

NOTE: Closing an Object purposely leaves that Object in whatever state was last commanded.

The host application can read capabilities or settings from the JSR Common API Library object or other JSR objects using one of these functions:

```
JSR_GetInfo(myObjectHandle …)       // Strings are in unicode
JSR_GetAsciiInfo(myObjectHandle …)  // Strings are ASCII
JSR_GetInt32(myObjectHandle …)      // Get one or an array of integer values
JSR_GetDouble(myObjectHandle …)     // Get one or an array of double values
JSR_GetString(myObjectHandle …)     // Strings are in unicode
JSR_GetAscii(myObjectHandle …)      // Strings are ASCII
```

The host application can control operation of the JSR Common API Library object or other JSR objects using one of these functions:

```
JSR_SetInt32(myObjectHandle …)      // Set one or an array of integer values
JSR_SetDouble(myObjectHandle …)     // Set one or an array of double values
```

The host application can call function `JSR_GetErrorJSRString()` or `JSR_GetErrorJSRAscii()` to decode numbered error messages to display the spelling of the enumerated status identifier. Your application does not need to contain strings to decode the status codes for display to the user. By using `JSR_GetErrorJSRString()` or `JSR_GetErrorJSRAscii()`, an application can operate with newer DLL's than it was originally written for and the DLL will properly decode error numbers that did not exist in the earlier version.

2.8 Using Objects

Before using any object a handle to the object must be available, and the object must be opened by the host application.

Finding an object handle:

- The handle of the Library is 1, hard coded as JSR_LIBRARY_HANDLE.
- Other handle values are dynamic and assigned at run-time.
- To determine the number of available handles,
  Call JSR_GetInfo(parentHandle, ID_of_desired_handle_property, &hostsJSRInfoStruct)
- Call JSR_GetInt32(parentHandle, ID_of_desired_handle_property, count, &handleArray) to get the array of handles.
- When you are certain only one instrument will be used, you may hard code the count of 1. If the instrument is not connected, the call to get the handle list will return an error status.
- Likewise, if you are using only PRC50 or DPR300 instruments, the count of channels, pulsers, and receivers may be hard-coded as 1.

Opening an Object:

All objects must be opened before any calls to set or get info or property values to/from that object.

The Library is opened via a call to JSR_OpenLibrary(…), all other objects are opened via a call to JSR_OpenObject(…) with the object handle as the first argument.

2.9 JSR Common API Data Types

In order to keep the number of interface functions small, there are only a small number of data types used in the API.

`typedef double JSR_Double;`
- The elementType is `JSR_TYPE_ENUM_DOUBLE`.
- Use `JSR_GetDouble()` and `JSR_SetDouble()`.
- The C double-precision “double” data type.
- One value occupies 8 bytes

`typedef unsigned short jsr_wchar;`
- 16- bit characters to support Unicode.
- This is the base class for JSR_String
- Single values are not supported by the API, just the array described below.

`#define JSR_SMALL_STRING_SIZE (64)`
`typedef jsr_wchar JSR_String[JSR_SMALL_STRING_SIZE];`
- The elementType is `JSR_TYPE_ENUM_STRING`.
- Use `JSR_GetString()` or `JSR_GetInfo()`
- Applications must use code for 16-bit characters

`typedef char JSR_Ascii[JSR_SMALL_STRING_SIZE];`
- The elementType is JSR_ JSR_TYPE_ENUM_STRING.
- Use `JSR_GetAscii()` or `JSR_GetAsciiInfo()`
- Standard 8-bit ASCII strings

`typedef signed int JSR_Int32`
- The elementType is `JSR_TYPE_ENUM_INT32`.
- Use `JSR_Get_Int32()` or `JSR_SetInt32()`
- Used for “values”
- This is the base type for the types listed below

`typedef signed int JSR_Enum`

- The elementType is `JSR_TYPE_ENUM_ENUM`.
- Use `JSR_Get_Int32()` or `JSR_SetInt32()`
- Used for selecting or identifying enumerated values
- A special case of `JSR_Int32`

`typedef signed int JSR_Handle`

- The elementType is `JSR_TYPE_ENUM_HANDLE`.
- Use `JSR_Get_Int32()` or `JSR_SetInt32()`
- Used for selecting or identifying a specific object handle
- A special case of `JSR_Int32`

`typedef signed int JSR_PropID`

- The elementType is `JSR_TYPE_ENUM_PROP_ID`.
- Use `JSR_Get_Int32()` or `JSR_SetInt32()`
- Used for selecting or identifying a specific property
- A special case of `JSR_Int32`

`typedef signed int JSR_Status`

- The elementType is `JSR_TYPE_ENUM_STATUS`.
- Use `JSR_Get_Int32()` or `JSR_SetInt32()`
- Used as a status or error code (see section 2.5.1)
- The "value" of a small number of properties
- A special case of `JSR_Int32`

`typedef signed int JSR_Bool`

- The elementType is `JSR_TYPE_ENUM_BOOL`.
- Use `JSR_Get_Int32()` or `JSR_SetInt32()`
- Used for false/true, disable/enable, off/on, etc
- False = 0, True = 1
- GUI could show as a check box, toggle switch, or 2-element menu
- A special case of `JSR_Int32`

2.9.1 Notes About JSR_Bool

Properties of elementType `JSR_TYPE_ENUM_BOOL` (data type `JSR_Bool`) will provide or accept only the values `JSR_FALSE` (zero) and `JSR_TRUE` (one).

Although in some contexts any non-zero variable is considered TRUE, the JSR Common API will accept only zero and one. If an application attempts to call JSR_SetInt32((a JSR_Bool property)) with a pointer to a value less than 0 or greater than 1, the JSR_SetInt32 function will return JSR_FAIL_VALUE_TOO_LOW or JSR_FAIL_VALUE_TOO_HIGH.

Read-only properties of type JSR_Bool can be represented on a GUI as an off/on LED, a disabled check box with 2 states, a single check mark that is present or not, or a non-editable text box with text indicating the state.

Read-write properties of type JSR_Bool can be represented on a GUI as an enabled check box, a pair of radio buttons, a toggle switch, a single button that toggles its caption, or a menu with two choices.

Most of the JSR_Bool properties in the API will have a non-zero info.ListID which will identify a string property appropriate to the controlled property, such as JSR_ID_ReferenceOffOnNames ("Off" or "On"), JSR_ID_ReferenceFalseTrueNames ("False" or "True"), or JSR_ID_ReferenceEnableNames ("Disabled" or "Enabled"). If the info.ListID is zero for a JSR_Bool property, your application must provide the default text of your choice if you use a GUI element that shows text. In some cases, you may want your application to display "0" or "1".

2.9.2 Notes on JSR_Status Status Codes

JSR_Status is used for managing status codes, from components as well as function returns. The status functions are broken down into a few separate sections for end developer ease of use. These status codes are usually mentioned by their enumeration names, however each enumeration represents a 32bit integer value.

The 'name' of status codes can be found by entering the status code into the `JSR_GetErrorJSRString()` or `JSR_GetErrorJSRAscii()` functions, and displaying the returned string.

The function `JSR_GetErrorJSRString()` or `JSR_GetErrorJSRAscii()` may be called even if the Library is not open (but the Library DLL must be loaded).

JSR_Status value	Description
0	This is the success code (JSR_OK). This indicates that everything is as expected. For function calls, this indicates no errors during the function.
1 - 1023	Reserved for application developer status codes. The JSR Common API does not return any errors in this range. These can be for error, warning, or debugging status codes. JSR_GetError functions will return a string stating that these are end developer error codes.
1024 - 2047	This section is reserved for warning codes. Warnings indicate that the function was successful (or the property is OK), but some form of minor error occurred which was recoverable/expected. JSR_GetError functions will return the warning's enum name.
2048 - 65,535	These values represent JSR Common API error codes. JSR_GetError functions will return the error's enum name.
65,536+	These error codes are reserved for future use. JSR_GetError functions will return a string that begins with the number then "'IS NOT A VALID JSR_LIB ERROR NUMBER"

2.10 How 'List' and 'Index' properties are linked

Some properties of pulsers and receivers have a small set of allowable values, such as a fixed set of damping resistors or a fixed set of high pass filters or low pass filters. The number of possible settings and the values those settings represent are not common among all JSR Pulser-Receivers, and so the application must read the list from the JSR Common API Library and control the value by setting the index value. In earlier JSR software, such settings were implemented using hard-coded enum values or #define's like "#define Damping100 1" indicating a control value of 1 would select the 100 ohm resistor.

For properties using the list and index mechanism, the JSR Common API Library provides two properties:

- A controllable "Index" property: always read-write and always of type JSR_Int32.
- A "List" property: always read-only and can be an array of type JSR_Int32, JSR_Double, or JSR_String.

For example:

```
double resistors[8]; // count is simplified for this example

// read the array of available damping resistor values
status = JSR_GetDouble(pulserHandle, JSR_ID_DampResistorList, 8, &resistors);

// select the second damping resistor value
JSR_Int32 resistorIndex = 1;
status = JSR_SetInt32(pulserHandle, JSR_ID_DampResistorIndex, 1, &resistorIndex)
```

The listID field of the JSR_InfoStruct for a property that is an Index will contain the enumerated JSR_PropertyID of the List associated with the Index. More than one Index property can refer to the same List property. Index values will always have zero as the lower limit. The host application should use the “limitHi” field of a property’s JSR_InfoStruct as the upper bound of the specific Index. The count of allowable values, and therefore the elementCount of the associated List will be (infoStruct.limitHi + 1)

2.11 Baseline Properties

Baseline properties are a subset of all properties. Baseline properties are common to all JSR instruments and provide access to the basic operation of all instruments.

Developers of host applications can safely hard-code access to baseline properties for one JSR instrument and be assured those baseline properties will operate the same way with other JSR instruments.

A specific baseline property may have a different element count, units, LimitLo, and LimitHi between different JSR instruments, but a given property will always have the same data type.

Note, for example, although every JSR instrument must have the JSR_ID_PulserDampResistorIndex property, each instrument could have a different number of resistors and/or different resistor values. Similarly, the LimitLo and LimitHi values for pulse repetition frequency are different for different instruments but the PRF is always controlled by the JSR_ID_PulserPRF property and is always of JSR_Double data type.

Some pulsers may have only one fixed value for a property where other pulsers allow several possible settings. In the case of a single, fixed, value, the property will still operate as a “read-write” property, but the SDK will accept only the one valid value. Other values will cause an error status return. In cases of a single valid value, the limitLo and limitHi elements of the property info struct will contain the same value.

2.12 Extended Properties

Different JSR instruments, e.g. PRC50 vs. DPR500 have different features and capabilities. In order to allow the JSR Common API to control the different features, each instrument family can have a different subset of extended features. For example, a PRC50 instrument will report its PCI slot number but a DPR500 will report its RS-232 serial comm. port number.

Developers of host applications can safely hard-code access to extended properties for one JSR instrument if there are no plans to use a different JSR instrument in the future.

However, if it is desired to make the host application operate with more than one model of JSR instrument, the JSR_GetInfo(…) or JSR_GetAsciiInfo(…) function can be used to provide flexibility.

Detailed information about each property of each object can be read by the host application by calling JSR_GetInfo(…) or JSR_GetAsciiInfo(…) for each property. That information about the property includes its data type, element count, limits, attributes (e.g. read/write), a text string of the property name, and a “displayText” description string which can be displayed to the user as a caption.

There are two ways to use this property information:

- A host application could have hard-coded controls for extended properties but still be backward compatible with other instruments with fewer properties by attempting to read each property but not creating that specific GUI element if JSR_GetInfo(…) returns JSR_FAIL_ID_NOT_VALID.

- Alternatively, the host application can read from each object the JSR_ID_AvailablePropertyIDs array of that specific object. The list will include both baseline and extended properties for that object. The host application then builds a GUI display or control by looping through each property from the array, calling

JSR_GetInfo(…) to read information about each property and building the appropriate display or control element at run time based on the information in each property's JSR_InfoStruct or JSR_AsciiInfoStruct.

Extended properties are always initialized with default values that allow the instrument to run in baseline mode without having the extended properties set by the host application.

2.13 Properties Not Yet Defined

The JSR Common API is easily extensible. JSR can add new properties for any library, instrument, channel, receiver, or pulser object by distributing a new JSR_CommonXXYY.dll. Host application code written for an older DLL will also work with the newer DLL.

JSR plans to add properties to the JSR Common API, to both the baseline and extended property sets, and we plan to use this same API for new products.

All properties follow a strict set of rules to provide a common interface and the JSR_InfoStruct as described in the Extended Properties section, above. By using information in the JSR_InfoStruct your application can create a GUI control for any future property even though you don't know what the property is at the time you write the code, since the data type, count, limits, and even a caption are in the InfoStruct.

Any properties added to the JSR Common API after shipment of version 1.0 will always be initialized with default values that allow the instrument to run in baseline mode without having the new properties read or controlled by the host application. However, the default operation will set the Pulser state to "disabled".

2.14 Simulation Mode

The developer of host application software can use the simulation mode to develop her/his own host application code to operate with a JSR Pulser-Receiver instrument before purchasing the particular JSR instrument. Similarly, several developers could be writing application code on several different computers even though only one instrument was purchased.

Simulation mode allows a developer to test her/his source code for compliance with the baseline and extended properties to ensure operation with several JSR instruments.

When the host application opens the JSR Common API Library, the host application may optionally specify "Simulation" mode in the inOptionBits argument by passing JSR_LIB_OPEN_OPTION_SIMULATE or JSR_LIB_OPEN_OPTION_SIMULATE_WITH_DRIVER. The JSR Common API will appear to connect to one instrument of each model supported by the library and underlying drivers and requested in the application's call to JSR_OpenLibrary(…). See JSR_OpenLibrary(…)

When either simulation mode is enabled, the JSR Common API Library will not actually attempt to find and connect with any physical JSR Pulser-Receiver instruments. Instead, the JSR Common API Library will simulate communication with instruments, channels, receivers, and pulsers. The JSR Common API Library will construct simulated software objects that exist only within the software.

Function calls to JSR_GetInfo(…), JSR_GetAsciiInfo(…), JSR_GetInt32(…), JSR_GetDouble(…), JSR_GetString(…), and JSR_GetAscii(…) will return values and strings as though actually connected to the specified physical instruments. The properties of the simulated objects will be just like the properties of the physical objects except for simulated serial numbers.

Function Calls to JSR_SetInt32(…) and JSR_SetDouble(…) will test the arguments for data type, count, limitLo, and limitHi, returning error messages as if the physical instrument was present. If the passed property value(s) pass all the tests, those values will be stored in memory within the JSR Common API Library for later function calls to "get" the properties, until the Library is closed.

The only difference an application will notice between a simulated instrument and a real JSR Pulser-Receiver instrument is the call of JSR_GetString(channelHandle, JSR_ID_ChannelDescription… or the equivalent JSR_GetAscii() will have the text "Sim " at the beginning of the returned string.

When in simulation mode, properties identified with the property attribute PROP_ATTRIB_VOLATILE will not change based on external stimulus, but the pulser "power limit" properties will operate as a function of the values of other properties, as they do for physical instruments.

The values of properties set by the application in simulation mode are not persistent between sessions.

2.15 Designing applications to allow drop-in upgrade of JSR_CommonXXYY.dll

The JSR Common API Library is designed to be universal for current and future JSR Pulser-Receiver instruments.

- Access to baseline features that are common to all JSR products is identical.
- Extended features use a very structured interface.
- Features not yet defined will be self-defining for a general host application GUI.
- A small number of function calls provide all current and future interfaces.
- New properties will default to operating values, and so may not need to be controlled.

The JSR Common API has been designed and implemented so additional JSR Pulser-Receiver product models, with different limits, capabilities, and options can work with the same interface. The guidelines in this section can help you design and code an application that can seamlessly work with several different JSR Pulser-Receiver products, whether the JSR product is internal and external to the host computer.

The JSR Common API has been designed and implemented so a newer version of JSR_CommonXXYY.dll dynamic library can be used by an older version of a host application with no change to the host application and no need to rebuild the host application.

Host application software should always test the JSR_Status value returned by every call to a function in the JSR Common API Library and properly respond to exceptions.

Host application software should not hard-code values that may change for different JSR Pulser-Receiver product models. Rather, values such as maximum and minimum pulse voltage should always be read from the product at run time by calling JSR_GetInfo() or JSR_GetAsciiInfo().

Host application software should be built by "#include" of files JSR_CommonTypes.h, JSR_Status.h, and JSR_PropertyID.h. Identifiers from those files should be used in host application source code, rather than hard-coded numeric values.

When linking with JSR_Common.Lib, host applications will also need to `#include "JSR_Common.h"` in modules that make calls to the JSR Common Library.

2.16 Debugging installation of your code.

We highly recommend that you install the JSR Control Panel in your development environment. The installer puts all necessary files into the correct folders and registers files and drivers.

Using JSR Control Panel can verify your hardware and connections (RS-232) are working and the drivers are installed correctly. You can see the properties that can be controlled, as well as the minimum, maximum, and default settings for each API property.

However, only one application may communicate with an instrument at one time. The second application will be given an error message or "No Instruments Found" message.

3 Installing 32 bit and 64 bit Applications

3.1 The easy way to install files and drivers on a target platform

The easiest way to install the proper files and drivers is to run the JSR Control Panel installer as it comes from Imaginant.

The JSR Control Panel installation wizard allows you to select which items to install or not install. The installer wizard will put all needed files into the correct folders and install the correct drivers.

A target platform that will not be used for software development does not need the C:\JSR_SDK\ folder or anything in it, so you should deselect the JSR SDK using the installer wizard.

If your application will control only DPR300's and DPR500's, but not PRC50's, during installation you can deselect installation of the PRC50 drivers and files. Likewise, if your application will control only PRC50's you can deselect DPR300's and DPR500's in the installer wizard.

The installer for the JSR Control Panel senses the OS configuration of the PC it is being installed on.

On a 32 bit OS, JSR_Common3232.dll and associated 32 bit files will be installed into

C:\Program Files\JSR Ultrasonics\JSR Control Panel\
and C:\Program Files\JSR Ultrasonics\PRC50\

On a 64 bit OS, the JSR Control Panel installer allows the user to choose whether to install the 32 bit version or the 64 bit version, or both. There is no functional difference between the 32bit and 64bit versions. The 64bit version is offered only as an example of a working 64bit application which uses the JSR Common Api.

Although both versions can be installed on a 64 bit OS platform for application development, we recommend you do that only if you are developing your own software on that platform. Existence of two versions on a PC will probably cause confusion for other operators, and only one application at a time will be able to allocate and control the instruments.

The 32 bit configuration (32 bit app on a 64 bit OS) will get installed in

C:\Program Files (x86)\JSR Ultrasonics\JSR Control Panel\

And C:\Program Files (x86)\JSR Ultrasonics\PRC50\

The 64 bit configuration (64 bit app on a 64 bit OS will get installed in

C:\Program Files (x86)\JSR Ultrasonics\JSR Control Panel\x64

3.2 If you decide to write your own installer

The JSR Common API Library and JSR Control Panel can run on the following operating systems:

Windows XP, 32 bit and 64 bit

Windows Embedded, 32 bit and 64 bit

Windows Vista, 32 bit and 64 bit

Windows 7, 32 bit and 64 bit

An older version is available for Windows 2000 which also runs on 32 bit Windows XP. If you prefer the older version, contact TechSupport@Imaginant.com

The 64 bit versions of Windows operating systems allow for optional operation of 32 bit Windows applications running under the “Windows on Windows64 (WoW64)” shell, as well as the native operation of true 64 bit applications. Those 32 bit applications that run under WoW64 will normally be installed in the c:\ProgramFiles(x86)\ folder.

The JSR Common API Library comes in 3 configurations (32:32, 32:64, and 64:64), with DLL’s, Lib files to link with, and PRC50 drivers specific to each configuration.

Which set of JSR Common API files you should use depends on the bit-depth of the OS it is running on and whether your application is compiled and linked as a 32 bit or 64 bit application.

The PRC50 / JSR Control Panel distribution CD contains these 3 configurations:

Your application	Windows OS	Configuration Name
32 bit	32 bit	32:32
32 bit	64 bit	32:64
64 bit	64 bit	64:64

The file names of some of the files are the same for the different configurations but the files are actually different, so great care must be exercised to copy the right files out of the specified folders within the C:\JSR_SDK folder and put them into the correct folders on the target platform. Tables further below provide that information.

The “driver” for the Imaginant PRC50 consists of several parts.

WinDriver, a multi-purpose PCI driver we purchased from Jungo, Inc. for redistribution:

Files Wdapi1021.dll, windrvr6.sys, and others

See : http://www.jungo.com/st/windriver_usb_pci_driver_development_software.html

Or http://www.jungo.com/ and navigate to “WinDriver”

Driver-related files specific to the PRC50 written by Imaginant:

Files PRC50.inf , JSR_CommonXXYY.dll, and others.

NOTE:

There are three versions of the file wdapi1021.dll. All three versions have the same filename. It is very important to put the files in the correct location. Please refer to the tables in the following three sections for information on which version of the wdapi1021.dll to use and where it should be copied to.

3.3 32:32 Configuration (32 bit app on 32 bit OS)

To install the PRC50 drivers on a 32 bit Windows OS, use the files in the C:\JSR_SDK\Drivers\x86 folder. The wdreg utilities will need to access the other files in this folder, such as the inf and sys files and others.

The utility program wdreg.exe is used to install drivers. To install the PRC50 driver requires the following command lines to be executed while in Adminstrator mode. To run a command line box, please go to the Accessories menu and right-click on the 'Command Prompt' icon, and then select 'Run As Administrator'.

```
wdreg -inf windrvr6.inf install
wdreg -inf prc50.inf install
```

The above actions will install the hardware driver. In addition to the hardware driver, the 32:32 version of the WinDriver api dll (Wdapi1021.dll) and the JSR Common API dll (JSR_Common3232.dll) must be placed in the C:\<windows>\system32 folder.

File Name	**Get It From C:\JSR_SDK\**	**Put It Into**
JSR_Common3232.dll	Bin	C:\<windows>\System32
JSR_Common3232.lib	Lib	Your Project Lib Folder (Note1)
Wdapi1021.dll	Drivers\x86\WinDriver3232	C:\<windows>\System32

Note 1: Needed only if you choose static linking of our DLL

3.4 32:64 Configuration (32 bit app on 64 bit OS)

To run 32 bit applications which communicate with the PRC50 on a 64 bit Windows operating system, it will be necessary to install the 64bit hardware drivers for the PRC50.

To install the PRC50 drivers on a 64 bit Windows OS, use the files in the C:\JSR_SDK\Drivers\x64 folder. The wdreg64.exe utility will need to access the other files in this folder.

The utility program wdreg64.exe is used to install 64bit PRC50 drivers. To install the PRC50 driver requires the following command lines to be executed while in Adminstrator mode. To run a command line box, please go to the Accessories menu and right-click on the 'Command Prompt' icon, and then select 'Run As Administrator'.

```
Wdreg64 -inf windrvr6.inf install
Wdreg64 -inf prc50.inf install
```

The above actions will install the hardware driver. In addition to the hardware driver, the 32:64 version of the WinDriver api dll (Wdapi1021.dll) and the JSR Common API dll (JSR_Common3264.dll) must be placed in the C:\<windows>\SysWOW64 folder.

NOTE:

32 bit applications can run on both 32 bit and 64 bit operating systems, if the 32:32 configuration install instructions are applied to 32 bit target systems, and the 32:64 Configuration instructions are applied to 64 bit target systems. 64bit applications will only run on 64 bit target systems where the 64:64 configuration install instructions have been followed.

File Name	Get It From C:\JSR_SDK\	Put It Into
JSR_Common3264.dll	Bin	C:\<windows>\SysWOW64
JSR_Common3264.lib	Lib	Your Project Lib Folder (Note1)
Wdapi1021.dll	Drivers\x86\WinDriver3264	C:\<windows>\SysWOW64

3.5 64:64 Configuration (64 bit app on 64 bit OS)

To run 64 bit applications which communicate with the PRC50 on a 64 bit Windows operating system, it will be necessary to install the 64bit hardware drivers for the PRC50.

To install the 64 bit drivers for the PRC50, please see the preceding section, since the same hardware driver is used for 32 bit applications and 64 applications.
For running 64bit applications which will use the PRC50, in addition to the hardware driver, the 64:64 version of the WinDriver api dll (Wdapi1021.dll) and the JSR Common API dll (JSR_Common6464.dll) must be placed in the C:\<windows>\System32 folder.

File Name	Get It From C:\JSR_SDK\	Put It Into
JSR_Common6464.dll	Bin	C:\<windows>\system32
JSR_Common6464.lib	Lib	Your Project Lib Folder (Note1)
Wdapi1021.dll	Drivers\x64\WinDriver6464	C:\<windows>\system32

4 Connecting your application to the DLL

Note: The alias JSR_COMMONXXYY is used in this document to refer to file name specific to a configuration: specifically:

JSR_Common3232.Lib, JSR_Common3264.Lib, or JSR_Common6464.Lib

JSR_Common3232.dll, JSR_Common3264.dll, or JSR_Common6464.dll

Depending on the needs of the host application, the JSR_CommonXXYY.dll Dynamic Link Library can be invoked by the host application using either of two methods. Sample code provided with the SDK shows both methods.

4.1 Link with JSR_CommonXXYY.Lib at project link time:

To use the simpler method of DLL library loading, the host application must #include "JSR_Common.h" where needed and link with JSR_CommonXXYY.Lib. When the Operating System launches the host application executable file, the Operating System (OS) will attempt to load JSR_CommonXXYY.dll.

If JSR_CommonXXYY.dll is not found in the appropriate path(s), the OS will display an error alert telling the name of the missing DLL. When the user dismisses the alert, the host application immediately exits. If the DLL was not found, no other portion of the host application can execute.

If this method is used, the Windows operating system will use the previously specified "Path" specification to search for the JSR_CommonXXYY.dll, looking first in the default directory.

If JSR_CommonXXYY.dll is loaded using the above method, your application should use the function calls exactly as specified in JSR_Common.h.

For example:

```
status = JSR_GetInt32(…);
```

4.2 Using JSR_Loader.c at run time:

If you choose to use the Microsoft Windows "LoadLibrary()" dynamic library loading, your source code must compile and link with JSR_Loader.c or a module you write that is similar to JSR_Loader.c. Within your host application, when the application is launched it should call JSR_LoaderLoadLib().

This method allows the host application full control over the file name and path to the JSR_CommonXXYY.dll, as the full path is an argument to the function JSR_LoaderLoadLib(…)

If JSR_LoaderLoadLib(…) does not find or properly load JSR_CommonXXYY.dll it will return an error status. Your host application can be programmed to optionally immediately exit, raise an alert, or ignore the error and continue the application without connecting to any JSR instrument.

If your application uses JSR_LoaderLoadLib(), it should **not** #include "JSR_Common.h" and should **not** link with JSR_CommonXXYY.Lib. Your application should use the indirect function calls as specified in the JSR_FunctionPtrs defined in JSR_Loader.h.

For example:

```
status = myJSRFuncs->GetInt32(…);
```

The JSR_Simple library accesses the JSR_Common library interface using this dynamic library loading method. The JSR_Common dynamic library is expected to be in the Windows system PATH or in the same directory as the application linked with the JSR_Simple library.

5 JSR Common API Library

5.1 Library Control

5.1.1 JSR_OpenLibrary()

```
JSR_LIB_API JSR_Status JSR_OpenLibrary
(
    JSR_LibOpenOptionsEnum    inOptionBits,     // Controls library behavior
    JSR_ModelEnum*            inLoadModelArray, // Instrument models to load
    JSR_Int32                 inArrayCount,     // number of elements in arg 2
    JSR_Int32                 inReserved0,      // Must be zero until defined
    JSR_Int32                 inReserved1       // Must be zero until defined
);
```

JSR_OpenLibrary() must be the first function call a host application makes to the JSR Common API Library. The arguments passed to the library in this call determine the behavior of the library to subsequent function calls. The state of these options can not be modified after the JSR Common API Library is opened.

JSR_OpenLibrary() allocates memory, and may initiate processing threads and instantiate other resources, so a host application should call JSR_CloseLibrary(…) before the application closes in order to properly release the memory and other resources in the correct order.

5.1.1.1 Parameters

JSR_LibOpenOptionsEnum inOptionBits

The default state of all options will always be specified as zero, so inOptionBits = 0 is always a valid value.

JSR may add additional options to the JSR Common API Library with any release version. Therefore a host application must have all bits that are not yet defined set to zero. If the JSR Common API Library detects any option bits are set that are not allowed by that version of the library, JSR_OpenLibrary() will return JSR_FAIL_LIB_INVALID_OPTION_BIT.

At the time this version of this document was prepared, the inOptionBits argument can have these values:

JSR_LibOpenOptionsEnum	Description
`JSR_LIB_OPEN_OPTION_DEFAULTS`	Opens the Library with all options at their default state, the JSR Common API will communicate with physical JSR instruments.
`JSR_LIB_OPEN_OPTION_SIMULATE`	JSR instrument products will be simulated in software, no underlying drivers or DLL's will be loaded, and no physical devices will connect. See the section above titled "Simulation".
`JSR_LIB_OPTION_SIM_WITH_DRIVER`	This option modifies the Simulate capability by also requiring the JSR Common API Library to load the underlying driver(s) and/or DLL's but does NOT actually connect to a physical instrument. This capability is useful for the host application to ensure the correct driver and/or DLL for the specified JSR instrument model(s) are properly installed. If none of the specified underlying drivers and/or DLL's are found, the call to JSR_OpenLibrary(…) will return an error status.
`JSR_LIB_OPTION_DISABLE_MUTEX`	The JSR Common DLL supports applications that use multiple threads to communicate with a JSR Instrument. An instrument can only process one command at a time, so the SDK has a mutex object (mutual exclusion) that will cause any of the Get and Set calls to the SDK to wait for any previous Get or Set calls to complete. The waiting thread uses an OS process call rather than eating up CPU time. The JSR_LIB_OPTION_DISABLE_MUTEX option allows an application to turn off the mutex **for testing**. If the JSR_LIB_OPTION_DISABLE_MUTEX option bit is set and a Set or Get call is made by a thread while the SDK is processing a Set or Get from a different thread, one of the status codes JSR_FAIL_MUTEX_ERROR or JSR_FAIL_MUTEX_TIMEOUT will be returned by the call made by the second thread. The JSR_LIB_OPTION_DISABLE_MUTEX bit will have no effect in single-threaded applications.

JSR_ModelEnum* inLoadModelArray

The inLoadModelArray argument is a pointer to an array of type JSR_ModelEnum. The array exists within the application and can be a single element.

The value(s) in inLoadModelArray allows the host application to control which model(s) of JSR instrument the JSR Common API Library will:

- Load underlying driver(s) and DLL's for and
- Connect to physical instruments OR simulate (depending on the inOptionBits argument)

The elements of the inLoadModelArray array can be the enum values for any of the instruments supported by the library:

```
JSR_MODEL_PRC50,
JSR_MODEL_DPR500,
JSR_MODEL_DPR300,
JSR_MODEL_ANY
```

A value of JSR_MODEL_ANY will cause the Library to attempt to load all underlying drivers and DLL's. JSR_MODEL_ANY is useful to allow your application to be written to work with future JSR products that conform to the JSR Common API, by just plugging in the new JSR_CommonXXYY.dll without the need to recompile the host application.

It is not an error if the host application provides an array that includes JSR_MODEL_ANY and any of the specific model enum values, but the Library will behave as if JSR_MODEL_ANY was the only value.

When JSR expands the JSR Common Library to operate with a new instrument, additional model enum values will be added to JSR_ModelEnum which can then be added specifically by an application. A host application written to open with a specific new instrument model using the explicit enum identifier for that new model will not be compatible with older versions of the DLL and the call to JSR_OpenLibrary() will return JSR_FAIL_LIB_MODEL_NOT_SUPPORTED. To avoid this problem, your host application should pass a pointer to an array or single value that contains the enum value JSR_MODEL_ANY, which will always be backward and forward compatible.

The call to JSR_OpenLibrary() will return JSR_OK if any one or more of the requested instrument drivers or DLL's for that instrument were found and one or more of the requested instruments were found.

The call to JSR_OpenLibrary() will return JSR_FAIL_LIB_COULD_NOT_OPEN if none of the requested drivers were found or if none of the requested instruments were found after finding the right driver or DLL.

If more than one instrument model was requested by the host application, the host application can determine which models of drivers and DLL's were actually found by:

```
JSR_PropInfo myInfo;
status = JSR_GetInfo(JSR_LIBRARY_HANDLE, JSR_ID_LibrarySupportedDrivers, &myInfo);

// test status
status = JSR_GetInt32(JSR_LIBRARY_HANDLE, JSR_ID_LibrarySupportedDrivers,
myInfo.elementCount, &driversFound);
```

If your host application is intended to work with only one specific JSR instrument model, then requesting only that model will ensure the JSR Common Library will not attempt to load drivers for, or attempt to connect with, other instruments.

The DPR500 and DPR300 instruments are serial RS-232 devices. If DPRIO3.DLL is installed in the same directory as JSR_CommonXXYY.dll and the call to JSR_OpenLibrary() includes models JSR_MODEL_ANY, JSR_MODEL_DPR500, or JSR_MODEL_DPR300, the library will attempt to allocate, connect to, and "ping" serial ports COM1 through COM8 searching for DPR500 or DPR300 instruments, whether such instruments are connected or not. The default timeout for each serial port connection failure is 500 milliseconds. To avoid the time delay caused by that search during JSR_OpenLibrary, the host application not designed to communicate with a DPR500 or DPR300 should either not request those instrument models or not have file DPRIO3.DLL in the same directory as JSR_CommonXXYY.dll.

<u>JSR_Int32 *inArrayCount*</u>

The *inArrayCount* argument passes the number of elements to expect in the array passed in by *inLoadModelArray.*

<u>JSR_Int32 *inReserved0*</u>

The inReserved0 argument to JSR_OpenLibrary() is provided to allow JSR to add features to the JSR_CommonXXYY.dll while maintaining compatibility with host applications written to work with earlier versions. Before inReserved0 is implemented and defined, the host application must pass the value zero as the argument. If the host software passes a value for the inReserved0 argument that is not valid, the call to JSR_OpenLibrary() will return JSR_FAIL_RESERVED_ARGUMENT_BAD. All future versions of the JSR Common API Library will allow the InReserved0 argument to have the value of zero.

<u>JSR_Int32 *inReserved1*</u>

The inReserved1 argument to JSR_OpenLibrary() is provided to allow JSR to add features to the JSR_CommonXXYY.dll while maintaining compatibility with host applications written to work with earlier versions. Before inReserved1 is implemented and defined, the host application must pass the value zero as the argument. If the host software passes a value for the inReserved1 argument that is not valid, the call to JSR_OpenLibrary() will return JSR_FAIL_RESERVED_ARGUMENT_BAD. All future versions of the JSR Common API Library will allow the InReserved1 argument to have the value of zero.

5.1.2 JSR_CloseLibrary()

```
JSR_Status JSR_CloseLibrary(void);
```

JSR_OpenLibrary() allocates memory, and may initiate processing threads and instantiate other resources, so a host application should call JSR_CloseLibrary() before the application closes in order to properly release the memory and other resources in the correct order.

Failure to close the library before unloading it can cause a run time crash.

When the application closes any parent object, the library will close all objects that are children of that parent, then close the parent:

- Closing a channel will close any receiver and pulser(s) it contains.
- Closing an instrument will close any channels it contains.

Closing the Library purposely leaves the instrument(s) and other Objects running in whatever state was last commanded to them. If your application must stop the generation of pulses when it exits, your application will need to specifically disable the pulser object(s) with the appropriate property before closing the object or Library.

5.1.3 JSR_GetErrorJSRString()

```
JSR_Status JSR_ JSR_GetErrorJSRString
(
  JSR_Status  inErrorNumber // status from some other JSR Lib function
  JSR_String* outPJSRStr    // Pointer to a JSR_String in application
);
```

JSR_GetErrorJSRString() returns Unicode (16-bit) character strings. If you want 8-bit characters, call JSR_GetErrorJSRAscii(). See next section.

JSR_GetErrorJSRString () operates whether the JSR Library is open or not, but the Library must be loaded.

JSR_GetErrorJSRString decodes a JSR_Status error number into a zero-terminated string representation of the inErrorNumber and the enum as spelled out in JSR_Status.h.

For example, if the value 1 is the input argument, the string created will be " 1 JSR_WARN_GENERAL "

JSR_GetErrorJSRString always fills in the string, even if the inErrorNumber argument is "JSR_OK". In that case, the string will be " 0 JSR_OK"

If the inErrorNumber is not valid, e.g. 4321, the string will display:
` " 4321 IS NOT A VALID JSR_LIB ERROR NUMBER"`

and the return value of the function will be `JSR_FAIL_ERROR_STRING_NOT_FOUND`.

To allow the application more flexibility, the returned string does not have "\n" on the end.

The numeric part is always formatted to have 5 digit places and a space. If you don't want your application to display the number, your application can use a pointer starting at index [6].

Example:

```
status = JSR_SomeFunction(...)
if ( status != JSR_OK )
{
    JSR_String errorStr;
    Status = JSR_GetErrorJSRString( status, errorStr);
    DisplayWchar(errorStr);
}
```

5.1.4 JSR_GetErrorJSRAscii()

```
JSR_Status JSR_ JSR_GetErrorJSRAscii
(
    JSR_Status  inErrorNumber // status from some other JSR Lib function
    JSR_Ascii*  outPJSRAscii  // Pointer to a JSR_Ascii string in application
);
```

JSR_GetErrorJSRAscii () returns 8-bit character strings but is otherwise identical to JSR_GetErrorString().

If you want Unicode (16-bit) character strings, call JSR_GetErrorJSRString(). See above.

JSR_GetErrorJSRAscii() operates whether the JSR Library is open or not, but the Library must be loaded.

5.1.5 JSR_OpenObject()

```
JSR_Status JSR_OpenObject
(
    JSR_Handle  inHandle,         // a Handle value acquired from the Library
    JSR_Int32   inOpenOptionBits // unused bits must be zero
);
```

JSR_OpenObject() causes the JSR Common API Library to establish a connection to the object specified by inHandle. Once an object is successfully opened, the application can perform JSR_GetInfo(), JSR_GetAsciiInfo(), and the several JSR_GetX() and JSR_SetX() function calls to read and write Properties within that object.

If there was an error connecting to the object, JSR_OpenObject() will return a status code indicating the error. If there was such an error, the object was not opened, so attempts to call other functions for this object will fail, and return the status code JSR_FAIL_OBJECT_NOT_OPEN.

5.1.6 JSR_CloseObject()

```
JSR_Status JSR_CloseObject
(
    JSR_Handle  inHandle      // handle of an open object
);
```

JSR_CloseObject() causes the JSR Common API Library to release a connection to an instrument object, a channel object, a receiver object, or a pulser object within the JSR Common API Library.

JSR_CloseObject() should normally be called by a host application working from the bottom to the top, in the order (receiver or pulser), channel, instrument, Library.

However, as a form of protection, a call by the application to JSR_CloseObject() will cause all objects that are children or grandchildren of that object to be closed before the specified object is closed. This is not an error condition, but attempts to access a child object after a parent object is closed will cause the return of the status code JSR_FAIL_OBJECT_NOT_OPEN.

Closing any object or the whole Library purposely leaves the instrument(s) and other Objects running in whatever state was last commanded to them. If your application must stop the generation of pulses when it exits, your application will need to specifically disable the pulser object(s) with the appropriate property before closing the object or Library.

5.2 Read and write of Object Properties

Each Library, instrument, channel, receiver, and pulser object carries its own collection of Properties (settings). Some Properties provide information such as identification and capabilities, and are therefore read-only. Other Properties provide control of the object from the host application and are therefore read-write. The host application will access Properties in objects of all levels using the same function calls to JSR_CommonXXYY.dll.

The first argument to these common property functions is the handle to the desired object. The handle for the JSR Common API Library object is always 1, but the handles to all other objects are allocated dynamically at run time, and may be different between sessions. The host application should not assume any sequence or relationship between object handles and should not treat handles as pointers or physical quantities.

The common functions to access object Properties are:

- `JSR_GetInfo()`
- `JSR_GetAsciiInfo()`
- `JSR_GetInt32()`
- `JSR_GetDouble()`
- `JSR_GetString()`
- `JSR_GetAscii()`
- `JSR_SetInt32()`
- `JSR_SetDouble()`

The host application specifies which particular Property of each object to access by the inPropertyID argument to those functions. The inPropertyID argument must be a valid Property enum value from JSR_PropertyID.h

Applications should `#include “JSR_PropertyID.h”`, and use the identifier for the desired Property. Never hard-code an integer constant as a Property identifier.

5.2.1 JSR_GetInfo()

```
JSR_Status JSR_GetInfo
(
  JSR_Handle      inHandle,   // library is 1, other handles are dynamic
  JSR_PropID      inID,       // enumerated ID of a Property
  JSR_InfoStruct* outInfo     // the DLL copies info into app’s memory
);
```

JSR_GetInfo() allows the host application to determine information about any Property of the JSR Common API Library or any Property of other objects. Note that this function does not provide the actual value of a Property, just information about the Property. This function does not need to be called for simple Properties, such as a single integer, a single double, or a single string. JSR_GetInfo is especially useful to determine the size of an array or the size of a list that can vary between different instruments, such as the number of available damping resistors, number of available filters, or number of energy settings.

The values within the JSR_InfoStruct returned by JSR_GetInfo() remain constant during a process. Therefore, an application can read the JSR_InfoStruct for just once per session for each specified Property and store the struct or the information within the application code.

Sting values of type JSR_String are returned as 16-bit (Unicode) characters. If your host application uses 8-bit characters, see the JSR_GetAsciiInfo() function definition in the next section.

JSR_GetInfo() fills in a JSR_InfoStruct in the application's memory:

```
typedef struct
{
    JSR_Int32       propertyID;           // ID value from JSR_PropertyID.h
    JSR_String      name;                 // Spelling as in JSR_PropertyID.h
    JSR_String      displayText;          // caption with spaces
    JSR_AttribEnum  attribs;              // e.g. write, list, volatile, etc
    JSR_Int32       listID;               // ID of associated list if not zero
    JSR_TypeEnum    elementType;          // int32, double, bool, enum, etc
    JSR_Int32       elementCount;         // number of elements
    JSR_Int32       byteCount;            // byte count of the property
    JSR_LimitUnion  limitLo;              // lower limit for writeable properties
    JSR_LimitUnion  limitHi;              // upper limit for writeable properties
    JSR_UnitsEnum   units;                // physical unit if not JSR_UNITS_NONE
} JSR_InfoStruct;
```

Field Name	**Constant/ Variable**	**Description**
propertyID	Constant	The property identifier value from JSR_PropertID.h
name	Constant	The spelling of the identifier of the Property in JSR_PropertyID.h. Zero terminated.
displayText	Variable	A description of the Property, zero terminated. E.g. "Pulse Repetition Frequency"
Attribs	Constant	The Property attributes (e.g. write, list, volatile, etc)
listID	Variable	If non-zero, this field has the Property identifier value of the list this Property is an index into. Most Properties do not use this field and it is therefore zero.

Properties that use listID	Value of *listID*
`JSR_ID_PulserDampResistorIndex`	`JSR_ID_PulserDampResistorList`
`JSR_ID_PulserEnergyIndex`	`JSR_ID_PulserEnergyList`
`JSR_ID_PulserPowerLimitEnergyIndex`	`JSR_ID_PulserEnergyList`
`JSR_ID_ReceiverLPFilterIndex`	`JSR_ID_ReceiverLPFilterList`
`JSR_ID_ReceiverHPFilterIndex`	`JSR_ID_ReceiverHPFilterList`

Field Name	**Constant/ Variable**	**Description**
elementType	Constant	The property type as defined by JSR_TypeEnum in JSR_Types.h
elementCount	Variable	The number of elements of the specified type. (Most commonly 1)
byteCount	Variable	The number of bytes, which is `(elementCount * sizeof(dataType)).` Both the count of elements and byte count are provided to allow the host application to use whichever is more convenient. The byte count can be used directly to allocate a buffer of info.byteCount bytes.
limitLo	Variable	The lower limit allowed for R/W Properties. This is zero for read-only properties.
limitHi	Variable	Upper limit for R/W Properties. This is zero for read-only properties.
units	Variable	The property units as defined by JSR_UnitsEnum in JSR_Types.h

For the special case of a JSR_String* Property, outInfo.elementCount will contain the number of strings (not characters), and is therefore is normally 1. The byteCount is the count of bytes allocated for the string. ByteCount is always the size of the allocation: a string with a single character will have the same byteCount as a sting with 63 characters.

The string has a terminating zero. An empty string is valid and will have a terminating zero as the first character.

5.2.2 JSR_GetAsciiInfo()

```
JSR_Status JSR_GetAsciiInfo
(
    JSR_Handle           inHandle,    // library is 1, other handles are dynamic
    JSR_PropID           inID,        // enumerated ID of a Property
    JSR_AsciiInfoStruct* outAsciiInfo // the DLL copies info into app’s memory
);
```

JSR_GetAsciiInfo() operates just like JSR_GetInfo() (see previous section) except the text strings within the JSR_AsciiInfo struct use 8-bit ASCII characters for host applications that do not need 16-bit characters. If your host application supports Unicode or other 16-bit character types, you do NOT want to use the JSR_GetAsciiInfo() and JSR_GetAscii() functions. JSR does not have a planned date to fully implement Unicode or other 16-bit characters, but will consider doing so based on customer demand.

The JSR_AsciiInfoStruct is just like JSR_InfoStruct except the two strings within the structs have different byte counts. Therefore the two structs are NOT interchangeable.

```
typedef struct
{
    JSR_Int32       propertyID;           // ID value from JSR_PropertyID.h
    JSR_Ascii       name;                 // Spelling as in JSR_PropertyID.h
    JSR_Ascii       displayText;          // caption with spaces
    JSR_AttribEnum  attribs;              // e.g. write, list, volatile, etc
    JSR_Int32       listID;               // ID of associated list if not zero
    JSR_TypeEnum    elementType;          // int32, double, bool, enum, etc
    JSR_Int32       elementCount;         // number of elements
    JSR_Int32       byteCount;            // byte count of the property
    JSR_LimitUnion  limitLo;              // lower limit for writeable properties
    JSR_LimitUnion  limitHi;              // upper limit for writeable properties
    JSR_UnitsEnum   units;                // physical unit if not JSR_UNITS_NONE
} JSR_AsciiInfoStruct;
```

5.2.3 JSR_GetInt32()

```
JSR_Status JSR_GetInt32
(
    JSR_Handle inHandle,      // library is 1, other handles are dynamic
    JSR_PropID inID,          // enumerated ID number of a Property
    JSR_Int32  inCount,       // count of elements application wants
    JSR_Int32* outpValue      // SDK copies data to app’s memory
);
```

JSR_GetInt32() allows the host application to get the value of any property of the JSR Common API Library or any open object within the library, whether a single value or an array of values, that has one of the data types that is a specific case of a 32 bit signed integer:

The only difference between properties of the different data types derived from signed int is how they are used:

Typedef	**Enum**	**Usage**
`JSR_Int32`	`JSR_TYPE_ENUM_INT32`	Integer number or index value
`JSR_Enum`	`JSR_TYPE_ENUM_ENUM`	Enumerated control value
`JSR_Bool`	`JSR_TYPE_ENUM_BOOL`	0=false, 1=true
`JSR_Handle`	`JSR_TYPE_ENUM_HANDLE`	Handle of an object
`JSR_PropID`	`JSR_TYPE_ENUM_PROP_ID`	Property identifier
`JSR_Status`	`JSR_TYPE_ENUM_STATUS`	Result of a function call or test

The host application always passes a pointer to a variable or a buffer that has been allocated within the memory space of the host application. The JSR_GetInt32() function will cause the JSR Common API Library to copy the value (or an array of values) into the host application's memory space.

The host application has control of how many pieces of information will be copied by the JSR Common API Library by setting the value of the inCount argument. The memory the host application allocated for outpValue must be large enough to hold inCount number of elements. The host application may control the inCount argument to get a subset of a list or array.

Strict type-checking is used. For a host application to get an array of values, the outpValue must be type-casted to "(JSR_Int32*)"

For example:

```
JSR_InfoStruct   resInfo;
Status = JSR_GetInfo(1, JSR_ID_SomeInt32Array, &resInfo);
JSR_Int32* myarray = new JSR_Int32[resInfo.elementCount];
Status = JSR_GetInt32(1, Some_JSR_ID, resInfo.elementCount, (JSR_Int32*) myarray);
```

5.2.4 JSR_GetDouble()

```
JSR_Status JSR_GetDouble
(
    JSR_Handle  inHandle,      // library is 1, other handles are dynamic
    JSR_PropID  inID,          // enumerated ID number of a Property
    JSR_Int32   inCount,       // count of elements app wants
    JSR_Double* outpValue      // SDK copies data to app's memory
);
```

JSR_GetDouble() allows the host application to get the value of a JSR_Double property from the JSR Common API Library or another open JSR object, whether the Property is a single value or an array of values.

The host application always passes a pointer to a variable or a buffer that has been allocated within the memory space of the host application. The JSR_GetDouble() function will cause the JSR Common API Library to copy the value (or array) into the host application's memory space.

The host application has control of how many pieces of information will be copied by the JSR Common API Library by setting the value of the inCount argument. The memory the host application allocated for outpValue must be large enough to hold inCount number of elements. The host application can control the inCount argument to get a subset of a list or array.

Strict type-checking is used. For a host application to get an array of values, the outpValue must be type-casted to JSR_Double*.

5.2.5 JSR_GetString()

```
JSR_Status JSR_GetString
(
    JSR_Handle  inHandle,  // library is 1, other handles are dynamic
    JSR_PropID  inID,      // enumerated ID number of a Property
    JSR_Int32   inCount,   // count of string arrays the app wants, not characters
    JSR_String* outpValue  // SDK copies data to app's memory
);
```

This function gets Unicode (16-bit) strings. Use JSR_GetAscii() to get 8-bit strings.

For most properties, inCount is one: the number of stings, not the number of characters.

JSR_GetString() allows the host application to get the value of a JSR_TYPE_ENUM_STRING, Property from the JSR Common API Library or an open JSR channel, whether the Property is a single string or an array of strings.

JSR_String is an array of JSR_STRING_SIZE number of characters. Even if you are expecting a shorter string, the host application must allocate a full-sized buffer.

The host application always passes a pointer to a variable or a buffer that has been allocated within the memory space of the host application. The JSR_GetString function will copy the string (or array of strings) into the host application's memory space.

The host application has control of how many pieces of information will be copied by the JSR_GetString function by controlling the value of the inCount argument. The memory the host application allocated for outpValue must be large enough to hold inCount number of JSR_String elements.

5.2.6 JSR_GetAscii()

```
JSR_Status JSR_GetString
(
    JSR_Handle  inHandle,  // library is 1, other handles are dynamic
    JSR_PropID  inID,      // enumerated ID number of a Property
    JSR_Int32   inCount,   // count of string arrays the app wants, not characters
    JSR_String* outpValue  // SDK copies data to app's memory
);
```

JSR_GetAscii is just like JSR_GetString() except for the character format.

This function gets 8-bit ASCII strings. Use JSR_GetString() to get Unicode (16-bit) strings.

For most properties, inCount is one: the number of stings, not the number of characters.

5.2.7 JSR_SetInt32()

```
JSR_Status JSR_SetInt32
(
    JSR_Handle inHandle,    // library is 1, other handles are dynamic
    JSR_PropID inID,        // enumerated ID number of a Property
    JSR_Int32  inCount,     // count of elements the application wants
    JSR_Int32* inpValue     // SDK copies data from app’s memory
);
```

JSR_SetInt32() allows the host application to set the value of a JSR_Int32, JSR_ENUM_ENUM, or JSR_Bool Property into the JSR Common API Library or an open JSR channel, whether the Property is a single value or an array of values.

The JSR_Int32 (signed long) is used to represent JSR_Int32, JSR_ENUM_ENUM, or JSR_Bool values.

The only difference between properties of the different data types derived from signed int is how they are used:

Typedef	Enum	Usage
`JSR_Int32`	`JSR_TYPE_ENUM_INT32`	Integer number or index
`JSR_Enum`	`JSR_TYPE_ENUM_ENUM`	Enumerated control value
`JSR_Bool`	`JSR_TYPE_ENUM_BOOL`	0=false, 1=true
`JSR_Handle`	`JSR_TYPE_ENUM_HANDLE`	Handle of an object
`JSR_PropID`	`JSR_TYPE_ENUM_PROP_ID`	Property identifier
`JSR_Status`	`JSR_TYPE_ENUM_STATUS`	Result of a function call or test

The host application always passes a pointer to a variable or a buffer that has been allocated within the memory space of the host application. The JSR_SetInt32() function will cause the JSR Common API Library to copy the value (or array) from the host application's memory space into memory within the JSR Common API Library.

The host application has control of how many pieces of information will be copied by the JSR Common API Library by setting the value of the inCount argument. The memory the host application allocated for inpValue must be large enough to hold inCount number of elements. The host application can control the inCount argument to set a subset of a list or array.

5.2.8 JSR_SetDouble()

```
JSR_Status JSR_SetDouble
(
    JSR_Handle inHandle,      // library is 1, other handles are dynamic
    JSR_PropID inID,          // enumerated ID number of a Property
    JSR_Int32  inCount,       // the count of elements application wants
    JSR_Double* inpValue      // SDK will copy data from application's mem
);
```

JSR_SetDouble() allows the host application to set the value of a JSR_Double Property into the JSR Common API Library or an open JSR channel, whether the Property is a single value or an array of values.

The host application always passes a pointer to a variable or a buffer that has been allocated within the memory space of the host application. The JSR_SetDouble() function will cause the JSR Common API Library to copy the value (or array) from the host application's memory space into memory within the JSR Common API Library.

The host application has control of how many pieces of information will be copied by the JSR Common API Library by setting the value of the inCount argument. The memory the host application allocated for inpValue must be large enough to hold inCount number of elements. The host application can control the inCount argument to set a subset of a list or array.

5.3 Pseudocode for a very simple Host Application

The following pseudocode is an example of host application calls to the JSR Common API Library:

The status return value of all calls to the JSR Library should be tested in the application. That testing is not shown here in order to keep this example simple. Status code checking is properly shown in the Sample Code provided as part of the SDK.

```
JSR_Status status = JSR_OK;

// Open all models available
JSR_Int32 modelArray[1] = {JSR_MODEL_ANY};
JSR_int32 modelCount = 1;

// Note: Status should really be checked after each function call

status = JSR_OpenLibrary(JSR_LIB_OPTION_DEFAULT, modelArray, modelCount, 0, 0);

JSR_Handle myInstrumentHandle = JSR_INVALID_HANDLE;
JSR_Handle myChannelHandle = JSR_INVALID_HANDLE;
JSR_Handle myReceiverHandle = JSR_INVALID_HANDLE;
JSR_Handle myPulserHandle = JSR_INVALID_HANDLE;

// Get a Intrument Handle and open it
status = JSR_GetInt32(JSR_LIBRARY_HANDLE,
    JSR_ID_LibraryInstrumentHandles, 1, &myInstrumentHandle);
status = JSR_OpenObject(myInstrumentHandle, JSR_INSTRUMENT_OPEN_DEFAULT);

// Get a Channel Handle and open it
status = JSR_GetInt32(myInstrumentHandle, JSR_ID_ChannelHandles, 1, &myChannelHandle);
status = JSR_OpenObject(myChannelHandle, JSR_CHANNEL_OPEN_DEFAULT);

// Get a Receiver Handle and open it
status = JSR_GetInt32(ChannelHandle, JSR_ID_ReceiverHandles, 1, &myReceiverHandle);
status = JSR_OpenObject(myReceiverHandle, JSR_RECEIVER_OPEN_DEFAULT);

// Get and Set Receiver Properties
status = JSR_GetType(ReceiverHandle, JSR_ID_ReceiverProperty, 1, &currentValue);
status = JSR_SetType(ReceiverHandle, JSR_ID_ReceiverProperty, 1, &newValue);

// Get a Pulser Handle and open it
status = JSR_GetInt32(ChannelHandle, JSR_ID_PulserHandles, 1, &myPulserHandle);
status = JSR_OpenObject(myPulserHandle, JSR_PULSER_OPEN_DEFAULT);

// Get and Set Pulser Properties
status = JSR_GetType(PulserHandle, JSR_ID_PulserProperty, &currentValue);
status = JSR_SetType(PulserHandle, JSR_ID_PulserProperty, &newValue);

// host application controls other components of the system

// Close Objects
status = JSR_CloseObject(myPulserHandle);
status = JSR_CloseObject(myReceiverHandle);
status = JSR_CloseObject(myChannelHandle);
status = JSR_CloseObject(myInstrumentHandle);

// Close Library
status = JSR_CloseLibrary();
```

For example, the following code will set the Pulse Repetition Frequency on a pulser to 4.3 KHz:

```
JSR_Double desiredPulseRepFreq = 4300.0;
status = JSR_SetDouble(myPulserHandle, JSR_ID_PulserPRF, 1, &desiredPulseRepFreq);
```

5.4 Using the JSR Common API Library Sample Code

The Sample Code provided as part of the SDK can be used for:

- Testing system and channel operation
- Examples of controlling specific features of the JSR Common API Library and Channels
- Providing a cut-and-paste source of working code for external developers.

The Sample Code provided as part of the SDK consist of:

- Source code
- Header files
- JSR_Loader.c source code
- Project definition files
- Executable code

Details for each piece of sample source code can be found within its documents.

The JSR Control Panel application distributed by JSR uses the JSR Common Library, so an application developer can use the JSR Control Panel to study the behavior of objects and properties within the JSR Common Library. Source code for the JSR Common Library will not be distributed as sample code.

6 JSR_Simple Library Interface

The JSR_Simple library interface was developed to simplify the use of the JSR_Common library. Property value operations are reduced to simple “Get” and “Set” methods. Property operations are performed on a specific channel. The current channel can be selected by using `SetChannel()`.

JSR_Simple inherits the same enumerations and property information structures from JSR_Common Library.

One differerence from the JSR_Common library is that the property values are passed by value for the “Set” methods.

String support for both JSR_String (Unicode) and JSR_Ascii is also provided. The interface automatically determines the string type based on the parameter type provided. JSR_String is recognized by the “BSTR“ parameter prototype and JSR_Ascii is recognized by the “char*” parameter prototype.

6.1 Class constructor / destructor

6.1.1 Constructors

Allocate class:

Prototype	*Static reference*
	`JSR_Simple ic;`
	Dynamic reference
	`JSR_Simple ic(JSR_LibOpenOptionsEnum, JSR_ModelEnum);`	

Call	*Static reference*
	`JSR_Simple ic;`
	Static reference opening library with parameters
	`JSR_Simple ic(JSR_LIB_OPTION_DEFAULT, JSR_MODEL_ANY);`
	Dynamic reference
	`JSR_Simple* ic = new JSR_Simple;`
	Dynamic reference opening library with parameters
	`JSR_Simple* ic = new JSR_Simple(JSR_LIB_OPTION_DEFAULT, JSR_MODEL_ANY);`	

Return	No return value from constructors.	**NOTE:** Check `GetLastError()`, `GetLoadLibraryStatus()` and `GetOpenLibraryStatus()` for successful operation.

6.1.2 Destructor

```
~JSR_Simple();
```

6.2 Library Functions

6.2.1 Open()

Open JSR_Common Library interface.

Prototype	`JSR_Status Open(JSR_LibOpenOptionsEnum, JSR_ModelEnum);`
Call	`JSR_Status status = ic.Open(JSR_LIB_OPTION_DEFAULT, JSR_MODEL_ANY);`
Return	`JSR_OK` if successful.

6.2.2 Close()

Close the JSR_Simple/JSR_Common interface:

Prototype	`JSR_Status Close();`
Call	`JSR_Status status = ic.close();`
Return	`JSR_OK` if successful.

6.3 Library methods

6.3.1 GetLoadLibraryStatus()

Get status from loading the JSR_Simple/JSR_Common interface.

Prototype	`JSR_Status GetLoadLibraryStatus();`
Call	`JSR_Status status = ic.GetLoadLibraryStatus();`
Return	JSR_OK if successful.

6.3.2 GetOpenLibraryStatus()

Get status from opening the JSR_Simple/JSR_Common interface channels.

Prototype	`JSR_Status GetOpenLibraryStatus();`
Call	`JSR_Status status = ic.GetOpenLibraryStatus();`
Return	JSR_OK if successful.

6.3.3 IsLoaded()

Check if JSR_Simple/JSR_Common interface is loaded.

Prototype	`JSR_Status IsLoaded();`
Call	`JSR_Status status = ic.IsLoaded();`
Return	JSR_OK if JSR_Simple/JSR_Common interface is loaded.

6.3.4 IsOpen()

Check if JSR_Simple/JSR_Common channels are open.

Prototype	`JSR_Status GetOpenLibraryStatus();`
Call	`JSR_Status status = ic.GetOpenLibraryStatus();`
Return	JSR_OK if JSR_Simple/JSR_Common interface is open.

6.3.5 GetSupportedModelCount()

Get the number of device models that are supported by the JSR_Simple/JSR_Common interface.

Prototype	`JSR_Int32 GetSupportedModelCount();`
Call	`JSR_Int32 count = ic.GetSupportedModelCount();`
Return	Number of supported models.

6.3.6 GetSupportedDriverCount()

Get the number of device drivers that are supported by the JSR_Simple/JSR_Common interface.

Prototype	`JSR_Int32 GetSupportedDriverCount();`
Call	`JSR_Int32 count = ic.GetSupportedDriverCount();`
Return	Number of supported drivers.

6.3.7 IsModelSupported()

Check if the supplied model enumerator is supported by the JSR_Simple/JSR_Common interface.

Prototype	`JSR_Bool IsModelSupported(JSR_ModelEnum eModel);`
Call	`JSR_Bool bSupported = ic.IsModelSupported(JSR_MODEL_PRC50);`
Return	JSR_TRUE if model is supported.

6.3.8 IsDriverSupported()

Check if the driver for the supplied model enumerator is supported by the JSR_Simple /JSR_Common interface.

Prototype	`JSR_Bool IsDriverSupported(JSR_ModelEnum eModel);`
Call	`JSR_Bool bSupported = ic.IsDriverSupported(JSR_MODEL_PRC50);`
Return	JSR_TRUE if driver for model is supported.

6.3.9 GetSupportedDrivers()

Get a string that contains the models supported by the JSR_Common interface.

Prototype	`BSTR GetSupportedDrivers();`
Call	`BSTR sSupported = ic.GetSupportedDrivers(JSR_MODEL_PRC50);`
Return	Unicode (16-bit) string containing the models supported.

6.3.10 GetModelName()

Get a string that contains the model name for the supplied model enumerator.

Prototype	`BSTR GetModelName(JSR_ModelEnum eModel);`
Call	`BSTR sName = ic.GetModelName(JSR_MODEL_PRC50);`
Return	Unicode (16-bit) string containing the model name.

6.3.11 GetUnitsName()

Get a string that contains the unit name for the supplied units enumerator.

Prototype	`BSTR GetUnitsName(JSR_UnitsEnum eUnits);`
Call	`BSTR sName = ic.GetUnitsName(JSR_UNITS_HERTZ);`
Return	Unicode (16-bit) string containing the units name.

6.3.12 GetDataTypeName()

Get a ascii or Unicode string that contains the data type name for the supplied type enumerator.

Prototype	*Unicode* `JSR_Status GetDataTypeName(JSR_TypeEnum ePropertyType, BSTR pVal);`
	Ascii `JSR_Status GetDataTypeName(JSR_TypeEnum ePropertyType, char* pVal);`

Call	*Unicode*
	`JSR_String jsrString;`
	`JSR_Status status = ic.GetDataTypeName(JSR_ID_PulserPRF, jsrString);`

	Ascii
	`JSR_Ascii jsrAscii;`
	`JSR_Status status = ic.GetDataTypeName(JSR_ID_PulserPRF, jsrAscii);`

Return	JSR_OK if successful. String parameter will contain the data type name.

6.4 Status and Diagnostic Methods

6.4.1 GetLastError()

Get the status of the last property operation:

Prototype	`JSR_Status GetLastError();`
Call	`JSR_Status status = ic.GetLastError();`
Return	The status of the last property operation.

6.4.2 GetLastErrorString()

Get the error string for the status of the last property accessed.

Prototype	*Unicode* `JSR_Status GetLastErrorString(BSTR pVal);`
	Ascii `JSR_Status GetLastErrorString(char* pVal);`

Call	*Unicode*
	`JSR_String jsrString;`
	`JSR_Status status = ic.GetLastErrorString(jsrString);`

	Ascii
	`JSR_Ascii jsrAscii;`
	`JSR_Status status = ic.GetLastErrorString(jsrAscii);`

Return	JSR_OK if successful. String parameter will contain the error message string.

6.4.3 GetErrorString()

Get the error string for the supplied status enumerator.

Prototype	*Unicode* `JSR_Status GetErrorString(JSR_Status status, BSTR);`
	Ascii `JSR_Status GetErrorString(JSR_Status status, char*);`

Call	*Unicode*
	`JSR_String jsrString;`
	`JSR_Status status = ic.GetErrorString(JSR_OK, jsrString);`

	Ascii
	`JSR_Ascii jsrAscii;`
	`JSR_Status status = ic.GetErrorString(JSR_OK, jsrAscii);`

Return	JSR_OK if successful. String parameter will contain the error message string.

6.4.4 LastProperty()

Get the property id of the last property operation: Useful for diagnostic purposes.

Prototype	`JSR_PropertyIDEnum LastProperty();`
Call	`JSR_PropertyIDEnum propid = ic.LastProperty();`
Return	The property id of the last property operation.

6.5 Channel Methods

6.5.1 GetChannelCount()

Returns the number of channels available via the JSR_Simple interface.

Prototype	`JSR_Int32 GetChannelCount();`
Call	`JSR_Int32 nCount = ic.GetChannelCount();`
Return	The number of channels available by the JSR_Common library.

6.5.2 GetChannel()

Get the current channel operated on by the JSR_Common library.

Prototype	`JSR_Int32 GetChannel();`
Call	`JSR_Int32 nChannel = ic.GetChannel();`
Return	The current channel or (–1) if no channel has been set.

6.5.3 SetChannel()

Set the current operating channel.

Prototype	`JSR_Status SetChannel(JSR_Int nChannel);`
Call	`JSR_Status status = ic.SetChannel(0);`
Return	JSR_OK if successful or JSR_FAIL_INSTRUMENT_NUMBER_NOT_VALID if channel is bad.

6.6 Base Property Methods

6.6.1 GetInt32()

Get the *Int32* value of the supplied property id.

Prototype	`JSR_Status GetInt32(JSR_PropertyIDEnum ePropId, JSR_Int32* pVal);`

Call	`JSR_Int32 nVal;`
	`JSR_Status status = ic.GetInt32(JSR_ID_PulserEnergyIndex, &nVal);`

Return	JSR_OK if successful. Integer property value is return by parameter reference.

6.6.2 GetDouble()

Get the *Double* value of the supplied property id.

Prototype	`JSR_Status GetDouble(JSR_PropertyIDEnum ePropId, JSR_Double* pVal);`

Call	`JSR_Double fVal;`
	`JSR_Status status =`
	`    ic.GetDouble(JSR_ID_PulserPowerLimitStatus, &fVal);`

Return	JSR_OK if successful. Double property value is return by parameter reference.

6.6.3 GetBool()

Get the *Bool* value of the supplied property id.

Prototype	`JSR_Status GetBool(JSR_PropertyIDEnum ePropId, JSR_BoolEnum* pVal);`

Call	`JSR_Bool bVal;`
	`JSR_Status status = ic.GetBool(JSR_ID_PulserTriggerEnable, &bVal);`

Return	JSR_OK if successful. Boolean property value is return by parameter reference.

6.6.4 GetStatus()

Get the *Status* value of the supplied property id.

Prototype	`JSR_Status GetStatus(JSR_PropertyIDEnum ePropId, JSR_Status* pVal);`

Call	`JSR_Status plStatus;`
	`JSR_Status status =`
	`    ic.GetStatus(JSR_ID_PulserPowerLimitStatus, &plStatus);`

Return	JSR_OK if successful. JSR_Status property value is return by parameter reference.

6.6.5 GetString()

Get the *String* value of the supplied property id.

Prototype	*Unicode* `JSR_Status GetString(JSR_PropertyIDEnum ePropId, char* pVal);`
	Ascii `JSR_Status GetString(JSR_PropertyIDEnum ePropId, BSTR pVal);`

Call	*Unicode*
	`JSR_String jsrString;`
	`JSR_Status status =`
	`    ic.GetString(JSR_ID_PulserTriggerEnable, jsrString);`

	Ascii
	`JSR_Ascii asciiString;`
	`JSR_Status status =`
	`   ic.GetString(JSR_ID_PulserTriggerEnable, asciiString);`

Return	JSR_OK if successful. String property value is return by parameter reference.

6.6.6 GetIndexValueInt32 ()

Get the *Int32* value of the property based on the current index value of the supplied property id.

Prototype	`JSR_Status GetIndexValueInt32(JSR_PropertyIDEnum ePropId, JSR_Int32* pVal);`
Call	`JSR_Int32 nVal;` `JSR_Status status = ic.GetIndexValueInt32(JSR_ID_PulserEnergyIndex, &nVal);`
Return	JSR_OK if successful. Integer property value is return by parameter reference.

6.6.7 GetIndexValueDouble ()

Get the *Double* of the property based on the current index value of the supplied property id.

Prototype	`JSR_Status GetIndexValueDouble(JSR_PropertyIDEnum ePropId, JSR_Double* pVal);`
Call	`JSR_Double fVal;` `JSR_Status status = ic.GetIndexValueDouble(JSR_ID_PulserPowerLimitStatus, &fVal);`
Return	JSR_OK if successful. Double property value is return by parameter reference.

6.6.8 GetIndexValueString ()

Get the *String* value of the property based on the current index value of the supplied property id.

Prototype	*Unicode* `JSR_Status GetIndexValueString(JSR_PropertyIDEnum ePropId, BSTR pVal);`
	Ascii `JSR_Status GetIndexValueString(JSR_PropertyIDEnum ePropId, char* pVal);`
Call	*Unicode* `JSR_String jsrString;` `JSR_Status status = ic.GetIndexValueString(JSR_ID_PulserTriggerEnable, jsrString);`
	Ascii `JSR_Ascii asciiString;` `JSR_Status status = ic.GetIndexValueString(JSR_ID_PulserTriggerEnable, asciiString);`
Return	JSR_OK if successful. String property value is return by parameter reference.

6.6.9 GetPropertyIndexValueInt32 ()

Get the *Int32* value of the property based on the supplied index value of the supplied property id.

Prototype	`JSR_Status GetPropertyIndexValueInt32(JSR_PropertyIDEnum ePropId, JSR_Int32 nIndex, JSR_Int32* pVal);`
Call	`JSR_Int32 nVal;` `JSR_Status status = ic.GetPropertyIndexValueInt32(JSR_ID_PulserEnergyIndex, 1, &nVal);`
Return	JSR_OK if successful. Integer property value is return by parameter reference.

6.6.10 GetPropertyIndexValueDouble ()

Get the *Double* value of the property based on the supplied index value of the supplied property id.

Prototype	`JSR_Status GetPropertyIndexValueDouble(JSR_PropertyIDEnum ePropId,`
	`    JSR_Int32 nIndex, JSR_Double* pVal);`

Call	`JSR_Double fVal;`
	`JSR_Status status =`
	`    ic.GetPropertyIndexValueDouble(JSR_ID_PulserPowerLimitStatus,`
	`    1, &fVal);`

Return	JSR_OK if successful. Double property value is return by parameter reference.

6.6.11 GetPropertyIndexValueString ()

Get the *String* value of the property based on the supplied index value of the supplied property id.

Prototype	*Unicode*
	`JSR_Status GetPropertyIndexValueString(JSR_PropertyIDEnum ePropId,`
	`    JSR_Int32 nIndex, BSTR pVal);`

	Ascii
	`JSR_Status GetPropertyIndexValueString(JSR_PropertyIDEnum ePropId,`
	`    JSR_Int32 nIndex, char* pVal);`

Call	*Unicode*
	`JSR_String jsrString;`
	`JSR_Status status =`
	`    ic.GetPropertyIndexValueString(JSR_ID_PulserTriggerEnable, 1,`
	`        jsrString);`

	Ascii
	`JSR_Ascii asciiString;`
	`JSR_Status status =`
	`   ic.GetPropertyIndexValueString(JSR_ID_PulserTriggerEnable, 1,`
	`       asciiString);`

Return	JSR_OK if successful. String property value is return by parameter reference.

6.6.12 GetPropertyMinValueInt32()

Get the Min *Int32* property value of the supplied property id.

Prototype	`JSR_Status GetPropertyMinValueInt32(JSR_PropertyIDEnum ePropId,`
	`    JSR_Int32* pVal);`

Call	`JSR_Int32 nVal;`
	`JSR_Status status =`
	`   ic.GetPropertyMinValueInt32(JSR_ID_PulserPowerLimitVolts, &nVal);`

Return	JSR_OK if successful. Integer property value is return by parameter reference.

6.6.13 GetPropertyMaxValueInt32()

Get the Max *Int32* property value of the supplied property id.

Prototype	`JSR_Status GetPropertyMaxValueInt32(JSR_PropertyIDEnum ePropId,`
	`    JSR_Int32* pVal);`

Call	`JSR_Int32 nVal;`
	`JSR_Status status =`
	`   ic.GetPropertyMaxValueInt32(JSR_ID_PulserPowerLimitVolts, &nVal);`

Return	JSR_OK if successful. Integer property value is return by parameter reference.

6.6.14 GetPropertyMinValueDouble()

Get the Min *Double* property value of the supplied property id.

Prototype	`JSR_Status GetPropertyMinValueDouble(JSR_PropertyIDEnum ePropId,`
	`    JSR_Double* pVal);`

Call	`JSR_Double fVal;`
	`JSR_Status status =`
	`   ic.GetPropertyMinValueDouble(JSR_ID_PulserPRF, &fVal);`

Return	JSR_OK if successful. Double property value is return by parameter reference.

6.6.15 GetPropertyMaxValueDouble()

Get the Max *Double* property value of the supplied property id.

Prototype	`JSR_Status GetPropertyMaxValueDouble(JSR_PropertyIDEnum ePropId,`
	`    JSR_Double* pVal);`

Call	`JSR_Double fVal;`
	`JSR_Status status =`
	`   ic.GetPropertyMaxValueDouble(JSR_ID_PulserPRF, &fVal);`

Return	JSR_OK if successful. Double property value is return by parameter reference.

6.6.16 GetInt32Array()

Get the array of *Int32* data for the supplied property id.

Prototype	`JSR_Status GetInt32Array(JSR_PropertyIDEnum ePropId,`
	`    JSR_Int32 nCount, JSR_Int32* pArray);`

Call	`JSR_Int32 nVal[5];`
	`JSR_Status status =`
	`   ic.GetInt32Array(JSR_ID_LibraryInstrumentHandles, 5, nVal);`

Return	JSR_OK if successful. Integer property values array is return by parameter reference.

6.6.17 GetDoubleArray()

Get the array of *Double* data for the supplied property id.

Prototype	`JSR_Status GetDoubleArray(JSR_PropertyIDEnum ePropId,`
	`    JSR_Int32 nCount, JSR_Double* pArray);`

Call	`JSR_Double fVal[5];`
	`JSR_Status status =`
	`   ic.GetDoubleArray(JSR_ID_PulserDampResistorList, 5, fVal);`

Return	JSR_OK if successful. Double property values array is return by parameter reference.

6.6.18 GetStringArray()

Get the array of *String* data for the supplied property id.

Prototype	*Unicode*
	`JSR_Status GetStringArray(JSR_PropertyIDEnum ePropId,`
	`    JSR_Int32 nCount, JSR_String* pArray);`

	Ascii
	`JSR_Status GetStringArray(JSR_PropertyIDEnum ePropId,`
	`    JSR_Int32 nCount, JSR_Ascii* pArray);`

Call	*Unicode*
	`JSR_String jsrString[4];`
	`JSR_Status status =`
	`    ic.GetStringArray(JSR_ID_ReferenceModelNames, 4, jsrString);`

	Ascii
	`JSR_Ascii asciiString[4];`
	`JSR_Status status =`
	`    ic.GetStringArray(JSR_ID_ReferenceModelNames, 4, asciiString);`

Return	JSR_OK if successful. The array of string values is returned by parameter reference.

The following allocates the string array that must be deallocated.

Prototype	*Unicode*
	`JSR_Status GetStringArray(JSR_PropertyIDEnum ePropId,`
	`    JSR_String* pArray);`

	Unicode
	`JSR_Status GetStringArray(JSR_PropertyIDEnum ePropId,`
	`    JSR_Ascii** pArray);`

Call	*Unicode*
	`JSR_String* jsrString;`
	`JSR_Status status =`
	`    ic.GetStringArray(JSR_ID_ReferenceModelNames, jsrString);`

	Unicode
	`JSR_Ascii* asciiString;`
	`JSR_Status status =`
	`    ic.GetStringArray(JSR_ID_ReferenceModelNames, asciiString);`

Return	JSR_OK if successful. A pointer to an allocated array of JSR_String values is return by parameter reference.

6.6.19 GetPropertyDisplayString()

Get the formatted string of the property value for the supplied property id.

Prototype	*Unicode*
	`JSR_Status GetPropertyDisplayString(JSR_PropertyIDEnum ePropId,`
	`    BSTR pVal);`

	Ascii
	`JSR_Status GetPropertyDisplayString(JSR_PropertyIDEnum ePropId,`
	`    char* pVal);`

Call	*Unicode*
	`JSR_String jsrString;`
	`JSR_Status status = ic.GetPropertyDisplayString(JSR_ID_PulserPRF,`
	`    jsrString);`

	Ascii
	`JSR_Ascii asciiString;`
	`JSR_Status status = ic.GetPropertyDisplayString(JSR_ID_PulserPRF,`
	`    asciiString);`

Return	JSR_OK if successful. String property value is set to parameter value.

6.6.20 FormatPropertyValue()

Return a formatted string based on the supplied property id and value.

Int32	
Prototype	*Unicode* `JSR_Status FormatPropertyValue(JSR_PropertyIDEnum ePropId, JSR_Int32 nVal, BSTR pVal);`
	Ascii `JSR_Status FormatPropertyValue(JSR_PropertyIDEnum ePropId, JSR_Int32 nVal, char* pVal);`
Call	*Unicode* `JSR_String jsrString;` `JSR_Status status = ic.FormatPropertyValue(JSR_ID_PulserTriggerEnable, 1, jsrString);`
	Ascii `JSR_Ascii asciiString;` `JSR_Status status = ic.FormatPropertyValue(JSR_ID_PulserTriggerEnable,1, asciiString);`
Return	JSR_OK if successful. String property value is return by parameter reference.
Double	
Prototype	*Unicode* `JSR_Status FormatPropertyValue(JSR_PropertyIDEnum ePropId, JSR_Double fVal, BSTR pVal);`
	Ascii `JSR_Status FormatPropertyValue(JSR_PropertyIDEnum ePropId, JSR_Double fVal, char* pVal);`
Call	*Unicode* `JSR_String jsrString;` `JSR_Status status = ic.FormatPropertyValue(JSR_ID_PulserPRF, 1234.5, jsrString);`
	Ascii `JSR_Ascii asciiString;` `JSR_Status status = ic.FormatPropertyValue(JSR_ID_PulserPRF, 1234.5, &asciiString);`
Return	JSR_OK if successful. String property value is return by parameter reference.

6.6.21 SetProperty()

Set value of property based on type supplied parameter datatype for the supplied property id.

Int32	
Prototype	`JSR_Status SetProperty(JSR_PropertyIDEnum ePropId, JSR_Int32 nValue);`

Call	`JSR_Status status =`
	`    ic.SetProperty(JSR_ID_PulserDampResistorIndex, 1);`

Return	JSR_OK if successful. Integer property value is is set to parameter value.
Double	

Prototype	`JSR_Status SetProperty(JSR_PropertyIDEnum ePropId,`
	`   JSR_Double fValue);`

Call	`JSR_Status status =`
	`   ic.SetProperty(JSR_ID_PulserPRF, 1234.5);`

Return	JSR_OK if successful. Double property value is is set to parameter value.
Status	

Prototype	`JSR_Status SetProperty(JSR_PropertyIDEnum ePropId,`
	`    JSR_Status nValue);`

Call	`JSR_Status status =`
	`   ic.SetProperty(JSR_ID_PulserPowerLimitStatus, (JSR_Status)JSR_OK);`

Return	JSR_OK if successful. Status property value is is set to parameter value.
Bool	
Prototype	`JSR_Status SetProperty(JSR_PropertyIDEnum ePropId, JSR_Bool nValue);`

Call	`JSR_Status status =`
	`   ic.SetProperty(JSR_ID_PulserTriggerEnable, (JSR_Bool)JSR_TRUE);`

Return	JSR_OK if successful. Bool property value is is set to parameter value.
String	
Prototype	*Unicode* `JSR_Status SetProperty(JSR_PropertyIDEnum ePropId, char* sValue);`
	Ascii `JSR_Status SetProperty(JSR_PropertyIDEnum ePropId, BSTR sValue);`
Call	*Unicode* `JSR_Status status = ic.SetProperty(JSR_ID_PulserPRF, “String”);`
	Ascii `JSR_Status status = ic.SetProperty(JSR_ID_PulserPRF, L“String”);`
Return	JSR_OK if successful. String property value is is set to parameter value.

6.6.22 SetInt32()

Set value of *Int32* property for the supplied property id.

Prototype	`JSR_Status SetInt32(JSR_PropertyIDEnum ePropId, JSR_Int32 nValue);`
Call	`JSR_Status status = ic.SetInt32(JSR_ID_PulserEnergyIndex, 1);`
Return	JSR_OK if successful. Integer property value is set to parameter value.

6.6.23 SetDouble()

Set value of *Double* property for the supplied property id.

Prototype	`JSR_Status SetDouble(JSR_PropertyIDEnum ePropId, JSR_Double fValue);`
Call	`JSR_Status status = ic.SetDouble(JSR_ID_PulserPRF, 1234.5);`
Return	JSR_OK if successful. Double property value is set to parameter value.

6.6.24 SetBool()

Set value of *Bool* property for the supplied property id.

Prototype	`JSR_Status SetBool(JSR_PropertyIDEnum ePropId, JSR_BoolEnum bValue);`
Call	`JSR_Status status = ic.SetBool(JSR_ID_PulserTriggerEnable, JSR_TRUE);`
Return	JSR_OK if successful. Boolean property value is set to parameter value.

6.6.25 SetString()

Set value of *String* property for the supplied property id only if that string is writable.

Prototype	*Unicode* `JSR_Status SetString(JSR_PropertyIDEnum ePropId, BSTR sValue);`
	Ascii `JSR_Status SetString(JSR_PropertyIDEnum ePropId, char* sValue);`
Call	*Unicode* `JSR_Status status = ic.SetString(JSR_ID_PulserPRF, L“String”);`
	Ascii `JSR_Status status = ic.SetString(JSR_ID_PulserPRF, “String”);`
Return	JSR_OK if successful. String property value is set to parameter value.

6.6.26 GetPropertyInfoName()

Get the property name for the supplied property id.

Prototype	*Unicode*
	`JSR_Status GetPropertyInfoName(JSR_PropertyIDEnum ePropId,`
	`    BSTR pVal);`

	Ascii
	`JSR_Status GetPropertyInfoName(JSR_PropertyIDEnum ePropId,`
	`    char* pVal);`

Call	*Unicode*
	`JSR_String jsrString;`
	`JSR_Status status = ic.GetPropertyInfoName(JSR_ID_PulserPRF,`
	`    jsrString);`

	Ascii
	`JSR_Ascii asciiString;`
	`JSR_Status status = ic.GetPropertyInfoName(JSR_ID_PulserPRF,`
	`    asciiString);`

Return	JSR_OK if successful. Name of property is set to parameter string value.

6.6.27 GetPropertyInfoDisplayName()

Get the property display name for the supplied property id.

Prototype	*Unicode* `JSR_Status GetPropertyInfoDisplayName(JSR_PropertyIDEnum ePropId, BSTR pVal);`
	Ascii `JSR_Status GetPropertyInfoDisplayName(JSR_PropertyIDEnum ePropId, char* pVal);`
Call	*Unicode* `JSR_String jsrString;` `JSR_Status status = ic.GetPropertyInfoDisplayName(JSR_ID_PulserPRF, jsrString);`
	Ascii `JSR_Ascii asciiString;` `JSR_Status status = ic.GetPropertyInfoDisplayName(JSR_ID_PulserPRF, asciiString);`
Return	JSR_OK if successful. Display name of property is set to parameter string value.

6.6.28 GetPropertyInfoDataType()

Get the property data type enumerator for the supplied property id.

Prototype	`JSR_Status GetPropertyInfoDataType(JSR_PropertyIDEnum ePropId, JSR_TypeEnum* pVal);`
Call	`JSR_TypeEnum dtype;` `JSR_Status status = ic.GetPropertyInfoDataType(JSR_ID_PulserPRF, &dtype);`
Return	JSR_OK if successful. JSR_TypeEnum value of property is return by parameter reference.

6.6.29 GetPropertyInfoDataTypeName()

Get the property data type name for the supplied property id.

Prototype	*Unicode* `JSR_Status GetPropertyInfoDataTypeName(JSR_PropertyIDEnum ePropId, BSTR pVal);`
	Ascii `JSR_Status GetPropertyInfoDataTypeName(JSR_PropertyIDEnum ePropId, char* pVal);`
Call	*Unicode* `JSR_String jsrString;` `JSR_Status status = ic.GetPropertyInfoDataTypeName(JSR_ID_PulserPRF, jsrString);`
	Ascii `JSR_Ascii asciiString;` `JSR_Status status = ic.GetPropertyInfoDataTypeName(JSR_ID_PulserPRF, asciiString);`
Return	JSR_OK if successful. Data type name of property is set to parameter string value.

6.6.30 GetPropertyInfoAttribute()

Get the property attribute enumerator for the supplied property id.

Prototype	`JSR_Status GetPropertyInfoAttribute(JSR_PropertyIDEnum ePropId,`
	`    JSR_AttribEnum* pVal);`

Call	`JSR_AttribEnum attr;`
	`JSR_Status status =`
	`    ic.GetPropertyInfoAttribute(JSR_ID_PulserPRF, &attr);`

Return	JSR_OK if successful. The value of property attribute is return by parameter reference.

6.6.31 GetPropertyInfoElementCount()

Get the property element count for the supplied property id.

Prototype	`JSR_Status GetPropertyInfoElementCount(JSR_PropertyIDEnum ePropId,`
	`    JSR_Int32* pVal);`

Call	`JSR_Int32 count;`
	`JSR_Status status =`
	`    ic.GetPropertyInfoAttribute(JSR_ID_PulserDampResistorList,`
	`        &count);`

Return	JSR_OK if successful. The count of element for the property is return by parameter reference.

6.6.32 IsPropertyAvailable()

Check if the property specified by supplied property id is available for the current channel.

Prototype	`JSR_Status IsPropertyAvailable(JSR_Handle handle,`
	`    JSR_PropertyIDEnum ePropId);`

Call	`JSR_Status status =`
	`   ic.IsPropertyAvailable(JSR_ID_PulserDampResistorList);`

Return	JSR_OK if property is available.

6.6.33 GetPropertyInfo ()

Retrieve property information structures for the supplied property id.

Prototype	*`JSR_InfoStruct`*
	`JSR_Status GetPropertyInfo(JSR_Handle handle,`
	`    JSR_PropertyIDEnum ePropId,`
	`    JSR_InfoStruct* pPropertyInfo);`

	`JSR_AsciiInfoStruct`
	`JSR_Status GetPropertyInfo(JSR_Handle handle,`
	`    JSR_PropertyIDEnum ePropId,`
	`    JSR_AsciiInfoStruct* pPropertyInfo);`

Call	*`JSR_InfoStruct`*
	`JSR_InfoStruct prop;`
	`JSR_Status status = ic.GetPropertyInfo(JSR_ID_PulserPRF, &prop);`

	`JSR_AsciiInfoStruct`
	`JSR_AsciiInfoStruct prop;`
	`JSR_Status status = ic.GetPropertyInfo(JSR_ID_PulserPRF, &prop);`

Return	JSR_OK if successful. The value of property attribute is return by parameter reference.

6.7 Handle Methods

You can perform operations on properties using handles obtained from the Handle Properties. Supply the handle before the Property Id parameter. These methods can be used for operations on properties on different channels.

The Base Property Methods ultimately call the Handle Methods by obtaining the Handle for you using the `GetHandle()` method on the supplied Property Id parameter.

6.7.1 Get Property Handle by Property

Get the handle based on the current channel and property id. This is the only channel dependent method in this section.

```
JSR_Handle GetHandle(JSR_PropertyIDEnum ePropId);
```

6.7.2 Get Property Value

Explicit property getters

JSR_Int32
```
JSR_Status GetInt32(JSR_Handle handle, JSR_PropertyIDEnum ePropId, JSR_Int32* pVal);
```

JSR_Double
```
JSR_Status GetDouble(JSR_Handle handle, JSR_PropertyIDEnum ePropId, JSR_Double* pVal);
```

JSR_Bool
```
JSR_Status GetBool(JSR_Handle handle, JSR_PropertyIDEnum ePropId, JSR_BoolEnum* pVal);
```

JSR_Status
```
JSR_Status GetStatus(JSR_Handle handle, JSR_PropertyIDEnum ePropId, JSR_Status* pVal);
```

JSR_String
```
JSR_Status GetString(JSR_Handle handle, JSR_PropertyIDEnum ePropId, BSTR pVal);
```

JSR_Ascii
```
JSR_Status GetString(JSR_Handle handle, JSR_PropertyIDEnum ePropId, char* pVal);
```

6.7.3 Set Property Value By Parameter Prototype

Set property values based on parameter prototypes.

JSR_Int32
```
JSR_Status SetProperty(JSR_Handle handle, JSR_PropertyIDEnum ePropId,
    JSR_Int32 nValue);
```
JSR_Double
```
JSR_Status SetProperty(JSR_Handle handle, JSR_PropertyIDEnum ePropId,
    JSR_Double fValue);
```
JSR_Bool
```
JSR_Status SetProperty(JSR_Handle handle, JSR_PropertyIDEnum ePropId,
    JSR_BoolEnum nValue);
```
JSR_Status
```
JSR_Status SetProperty(JSR_Handle handle, JSR_PropertyIDEnum ePropId,
    JSR_Status nValue);
```
JSR_String
```
JSR_Status SetProperty(JSR_Handle handle, JSR_PropertyIDEnum ePropId,
    BSTR sValue);
```
JSR_Ascii
```
JSR_Status SetProperty(JSR_Handle handle, JSR_PropertyIDEnum ePropId,
    char* sValue);
```

6.7.4 Set Property Value By Explicit Type

Set property values based on parameter prototypes.

JSR_Int32
```
JSR_Status SetInt32(JSR_Handle handle, JSR_PropertyIDEnum ePropId,
    JSR_Int32 nValue);
```
JSR_Double
```
JSR_Status SetDouble(JSR_Handle handle, JSR_PropertyIDEnum ePropId,
    JSR_Double fValue);
```
JSR_Bool
```
JSR_Status SetBool(JSR_Handle handle, JSR_PropertyIDEnum ePropId,
    JSR_BoolEnum nValue);
```
JSR_Status
```
JSR_Status SetStatus(JSR_Handle handle, JSR_PropertyIDEnum ePropId,
    JSR_Status nValue);
```
JSR_String
```
JSR_Status SetString(JSR_Handle handle, JSR_PropertyIDEnum ePropId,
    BSTR sValue);
```
JSR_Ascii
```
JSR_Status SetString(JSR_Handle handle, JSR_PropertyIDEnum ePropId,
    char* sValue);
```

6.7.5 Get Property Value Based on Current Index Value

Get the value of the current index for an index property.

```
JSR_Status GetIndexValueInt32(JSR_Handle handle, JSR_PropertyIDEnum ePropId,
    JSR_Int32* pVal);
JSR_Status GetIndexValueDouble(JSR_Handle handle, JSR_PropertyIDEnum ePropId,
    JSR_Double* pVal);
JSR_Status GetIndexValueString(JSR_Handle handle, JSR_PropertyIDEnum ePropId,
    BSTR pVal);
JSR_Status GetIndexValueString(JSR_Handle handle, JSR_PropertyIDEnum ePropId,
    char* pVal);
```

6.7.6 Get Property Value Based on Given Index Value

Get the value of a given index for an index property.

```
JSR_Status GetPropertyIndexValueInt32(JSR_Handle handle,
    JSR_PropertyIDEnum ePropId, JSR_Int32 nIndex, JSR_Int32* pVal);
JSR_Status GetPropertyIndexValueDouble(JSR_Handle handle,
    JSR_PropertyIDEnum ePropId, JSR_Int32 nIndex, JSR_Double* pVal);
JSR_Status GetPropertyIndexValueString(JSR_Handle handle,
    JSR_PropertyIDEnum ePropId, JSR_Int32 nIndex, BSTR pVal);
JSR_Status GetPropertyIndexValueString(JSR_Handle handle,
    JSR_PropertyIDEnum ePropId, JSR_Int32 nIndex, char* pVal);
```

6.7.7 Get Min and Max Property Values

Get minimum and maximum values of property.

```
JSR_Status GetPropertyMinValueInt32(JSR_Handle handle,
    JSR_PropertyIDEnum ePropId, JSR_Int32* pVal);
JSR_Status GetPropertyMinValueDouble(JSR_Handle handle,
    JSR_PropertyIDEnum ePropId, JSR_Double* pVal);
JSR_Status GetPropertyMaxValueInt32(JSR_Handle handle,
    JSR_PropertyIDEnum ePropId, JSR_Int32* pVal);
JSR_Status GetPropertyMaxValueDouble(JSR_Handle handle,
    JSR_PropertyIDEnum ePropId, JSR_Double* pVal);
```

6.7.8 Get Array Data from Property

Get an array of values from a property.

```
JSR_Status GetInt32Array(JSR_Handle handle, JSR_PropertyIDEnum ePropId,
    JSR_Int32 nCount, JSR_Int32* pArray);
```

```
JSR_Status GetDoubleArray(JSR_Handle handle, JSR_PropertyIDEnum ePropId,
    JSR_Int32 nCount, JSR_Double* pArray);
```

```
JSR_Status GetStringArray(JSR_Handle handle, JSR_PropertyIDEnum ePropId,
    JSR_Int32 nCount, JSR_String* pArray);
```

```
JSR_Status GetStringArray(JSR_Handle handle, JSR_PropertyIDEnum ePropId,
    JSR_Int32 nCount, JSR_Ascii* pArray);
```

The following allocates the string array that must be deallocated.

```
JSR_Status GetStringArray(JSR_Handle handle, JSR_PropertyIDEnum ePropId,
   JSR_String** pArray);
```

```
JSR_Status GetStringArray(JSR_Handle handle, JSR_PropertyIDEnum ePropId,
   JSR_Ascii** pArray);
```

6.7.9 Get Property Information

Get property information from a property:

- Property availability
- JSR_InfoStruct
- JSR_AsciiInfoStuct
- Name
- Display Name
- Data Type
- Data Type Name
- Property Attributes
- Element Count

```
JSR_Status IsPropertyAvailable(JSR_Handle handle, JSR_PropertyIDEnum ePropId);

JSR_Status GetPropertyInfo(JSR_Handle handle, JSR_PropertyIDEnum ePropId,
    JSR_InfoStruct* pPropertyInfo);

JSR_Status GetPropertyInfo(JSR_Handle handle, JSR_PropertyIDEnum ePropId,
    JSR_AsciiInfoStruct* pPropertyInfo);

JSR_Status GetPropertyInfoDisplayName(JSR_Handle handle,
    JSR_PropertyIDEnum ePropId, char* pVal);

JSR_Status GetPropertyInfoDisplayName(JSR_Handle handle,
    JSR_PropertyIDEnum ePropId, BSTR pVal);

JSR_Status GetPropertyInfoName(JSR_Handle handle, JSR_PropertyIDEnum ePropId,
    char* pVal);

JSR_Status GetPropertyInfoName(JSR_Handle handle, JSR_PropertyIDEnum ePropId,
    BSTR pVal);

JSR_Status GetPropertyInfoDataType(JSR_Handle handle, JSR_PropertyIDEnum ePropId,
    JSR_TypeEnum* pVal);

JSR_Status GetPropertyInfoDataTypeName(JSR_Handle handle,
    JSR_PropertyIDEnum ePropId, char* pVal);

JSR_Status GetPropertyInfoDataTypeName(JSR_Handle handle,
    JSR_PropertyIDEnum ePropId, BSTR pVal);

JSR_Status GetPropertyInfoAttribute(JSR_Handle handle, JSR_PropertyIDEnum ePropId,
    JSR_AttribEnum* pVal);

JSR_Status GetPropertyInfoElementCount(JSR_Handle handle,
    JSR_PropertyIDEnum ePropId, JSR_Int32* pVal);
```

6.7.10 Format Given Values by Property

Format given values based on a specified property id.

```
JSR_Status FormatPropertyValue(JSR_Handle handle, JSR_PropertyIDEnum ePropId,
    JSR_Int32 nVal, BSTR pVal);

JSR_Status FormatPropertyValue(JSR_Handle handle, JSR_PropertyIDEnum ePropId,
    JSR_Double fVal, BSTR pVal);

JSR_Status FormatPropertyValue(JSR_Handle handle, JSR_PropertyIDEnum ePropId,
    JSR_Int32 nVal, char* pVal);

JSR_Status FormatPropertyValue(JSR_Handle handle, JSR_PropertyIDEnum ePropId,
    JSR_Double fVal, char* pVal);
```

6.7.11 Format Current Property Value

Format the current value of a property.

```
JSR_Status GetPropertyDisplayString(JSR_Handle handle, JSR_PropertyIDEnum ePropId,
    char* pVal);
JSR_Status GetPropertyDisplayString(JSR_Handle handle, JSR_PropertyIDEnum ePropId,
    BSTR pVal);
```

6.8 Convenience Methods For Properties

These methods are all built upon the Base Property Methods. You do not specify the Property Id in the calling interface. All convienence methods operate on the currently set channel.

6.8.1 JSR_ID_ChannelDescription

6.8.1.1 GetChannelDescriptionString()

Get the description for the current channel.

Prototype	*Unicode* `JSR_Status GetChannelDescriptionString(BSTR pString);`
	Ascii `JSR_Status GetChannelDescriptionString(char* pString);`
Base	`GetString(JSR_ID_ChannelDescription, pString);`

Call	*Unicode*
	`JSR_String jsrString;`
	`JSR_Status status = ic.GetChannelDescriptionString(jsrString);`

	Ascii
	`JSR_Ascii asciiString;`
	`JSR_Status status = ic.GetChannelDescriptionString(asciiString);`

Return	JSR_OK if successful. Description of current channel is set to parameter string value.

6.8.2 JSR_ID_ChannelLetter

6.8.2.1 GetChannelLetter()

Get the channel letter for the current channel.

Prototype	*Unicode* `JSR_Status GetChannelLetter(BSTR pString);`
	Ascii `JSR_Status GetChannelLetter(char* pString);`
Base	`GetString(JSR_ID_ChannelLetter, pString);`

Call	*Unicode*
	`JSR_String jsrString;`
	`JSR_Status status = ic.GetChannelLetter(jsrString);`

	Ascii
	`JSR_Ascii asciiString;`
	`JSR_Status status = ic.GetChannelLetter(asciiString);`

Return	JSR_OK if successful. String containing the channel letter of current channel is set to parameter string value.

6.8.3 JSR_ID_PulserTriggerEnable

6.8.3.1 GetPulserTriggerEnable()

Get the trigger enable state for the current channel.

Prototype	`JSR_Status GetPulserTriggerEnable(JSR_BoolEnum* pVal);`
Base	`GetBool(JSR_ID_PulserTriggerEnable, (JSR_BoolEnum*)pVal);`

Call	`JSR_BoolEnum state;`
	`JSR_Status status = ic.GetPulserTriggerEnable(&state);`

Return	JSR_OK if successful. The Boolean property value is return by parameter reference.

6.8.3.2 SetPulserTriggerEnable()

Set the trigger enable state for the current channel.

Prototype	`JSR_Status SetPulserTriggerEnable(JSR_BoolEnum bVal);`
Base	`SetBool(JSR_ID_PulserTriggerEnable, bVal);`
Call	`JSR_Status status = ic.SetPulserTriggerEnable(JSR_TRUE);`
Return	JSR_OK if successful.

6.8.3.3 GetPulserTriggerEnableString()

Get the pulser trigger enable state as a string value for the current channel.

Prototype	*Unicode* `JSR_Status GetPulserTriggerEnableString(BSTR pString);`
	Ascii `JSR_Status GetPulserTriggerEnableString(char* pString);`
Base	`GetPropertyDisplayString(JSR_ID_PulserTriggerEnable, pString);`

Call	*Unicode*
	`JSR_String jsrString;`
	`JSR_Status status = ic.GetPulserTriggerEnableString(jsrString);`

	Ascii
	`JSR_Ascii asciiString;`
	`JSR_Status status = ic.GetPulserTriggerEnableString(asciiString);`

Return	JSR_OK if successful. The Boolean string value of trigger enable is set to parameter string value.

6.8.4 JSR_ID_PulserTriggerSource

6.8.4.1 GetPulserTriggerSource()

Get the trigger enable state for the current channel.

Prototype	`JSR_Status GetPulserTriggerSource(JSR_TriggerSourceEnum* pVal);`
Base	`GetInt32(JSR_ID_PulserTriggerSource, (JSR_Int32*)pVal);`

Call	`JSR_TriggerSourceEnum source;`
	`JSR_Status status = ic.GetPulserTriggerSource(&source);`

Return	JSR_OK if successful. The trigger source value is return by parameter reference.

6.8.4.2 SetPulserTriggerSource()

Set the trigger source for the current channel.

Prototype	`JSR_Status SetPulserTriggerSource(JSR_TriggerSourceEnum eVal);`
Base	`SetInt32(JSR_ID_PulserTriggerSource, (JSR_Int32)eVal);`
Call	`JSR_Status status = ic.SetPulserTriggerSource(JSR_TRIGGER_INTERNAL);`
Return	JSR_OK if successful.

6.8.4.3 GetPulserTriggerSourceString()

Get the trigger source as a string value for the current channel.

Prototype	*Unicode* `JSR_Status GetPulserTriggerSourceString(BSTR pString);`
	Ascii `JSR_Status GetPulserTriggerSourceString(char* pString);`
Base	`GetPropertyDisplayString(JSR_ID_PulserTriggerSource, pString);`

Call	*Unicode*
	`JSR_String jsrString;`
	`JSR_Status status = ic.GetPulserTriggerEnableString(jsrString);`

	Ascii
	`JSR_Ascii asciiString;`
	`JSR_Status status = ic.GetPulserTriggerEnableString(asciiString);`

Return	JSR_OK if successful. The Boolean string value of trigger enable is set to parameter string value.

6.8.5 JSR_ID_PulserPRF

6.8.5.1 GetPulserPRF()

Get the PRF for the current channel.

Prototype	`JSR_Status GetPulserPRF(JSR_Double* pVal);`
Base	`GetDouble(JSR_ID_PulserPRF, pVal);`

Call	`JSR_Double fPRF;`
	`JSR_Status status = ic.GetPulserTriggerSource(&fPRF);`

Return	JSR_OK if successful. The pulser PRF value is return by parameter reference.

6.8.5.2 SetPulserPRF()

Set the PRF for the current channel.

Prototype	`JSR_Status SetPulserPRF(JSR_Double fVal);`
Base	`SetDouble(JSR_ID_PulserPRF, fVal);`
Call	`JSR_Status status = ic.SetPulserPRF(1234.5);`
Return	JSR_OK if successful.

6.8.5.3 GetPulserPRFMin()

Get the Minimum PRF value for the current channel.

Prototype	`JSR_Status GetPulserPRFMin(JSR_Double* pVal);`
Base	`GetPropertyMinValueDouble(JSR_ID_PulserPRF, pVal);`

Call	`JSR_Double fPRF;`
	`JSR_Status status = ic.GetPulserPRFMin(&fPRF);`

Return	JSR_OK if successful. The minimum pulser PRF value is return by parameter reference.

6.8.5.4 GetPulserPRFMax()

Get the Minimum PRF value for the current channel.

Prototype	`JSR_Status GetPulserPRFMax(JSR_Double* pVal);`
Base	`GetPropertyMaxValueDouble(JSR_ID_PulserPRF, pVal);`

Call	`JSR_Double fPRF;`
	`JSR_Status status = ic.GetPulserPRFMax(&fPRF);`

Return	JSR_OK if successful. The maximum pulser PRF value is return by parameter reference.

6.8.5.5 GetPulserPRFString()

Get the pulser PRF as a string value for the current channel.

Prototype	*Unicode* `JSR_Status GetPulserPRFString(BSTR pString);`
	Ascii `JSR_Status GetPulserPRFString(char* pString);`
Base	`GetPropertyDisplayString(JSR_ID_PulserPRF, pString);`

Call	*Unicode*
	`JSR_String jsrString;`
	`JSR_Status status = ic.GetPulserPRFString(jsrString);`

	Ascii
	`JSR_Ascii asciiString;`
	`JSR_Status status = ic.GetPulserPRFString(asciiString);`

Return	JSR_OK if successful. The pulser PRF string value is set to parameter string value.

6.8.6 JSR_ID_PulserVolts

6.8.6.1 GetPulserVolts()

Get the pulser volts for the current channel.

Prototype	`JSR_Status GetPulserVolts(JSR_Double* pVal);`
Base	`GetDouble(JSR_ID_PulserVolts, pVal);`

Call	`JSR_Double fVolts;`
	`JSR_Status status = ic.GetPulserVolts(&fVolts);`

Return	JSR_OK if successful. The pulser volts value is return by parameter reference.

6.8.6.2 SetPulserVolts()

Set the pulser Volts for the current channel.

Prototype	`JSR_Status SetPulserVolts(JSR_Double fVal);`
Base	`SetDouble(JSR_ID_PulserVolts, fVal);`
Call	`JSR_Status status = ic.SetPulserVolts(110.0);`
Return	JSR_OK if successful.

6.8.6.3 GetPulserVoltsMin()

Get the Minimum PRF value for the current channel.

Prototype	`JSR_Status GetPulserVoltsMin(JSR_Double* pVal);`
Base	`GetPropertyMinValueDouble(JSR_ID_PulserVolts, pVal);`

Call	`JSR_Double fVolts;`
	`JSR_Status status = ic.GetPulserVoltsMin(&fVolts);`

Return	JSR_OK if successful. The minimum pulser Volts value is return by parameter reference.

6.8.6.4 GetPulserVoltsMax()

Get the Minimum PRF value for the current channel.

Prototype	`JSR_Status GetPulserVoltsMax(JSR_Double* pVal);`
Base	`GetPropertyMaxValueDouble(JSR_ID_PulserVolts, pVal);`

Call	`JSR_Double fVolts;`
	`JSR_Status status = ic.GetPulserVoltsMax(&fVolts);`

Return	JSR_OK if successful. The maximum pulser Volts value is return by parameter reference.

6.8.6.5 GetPulserVoltsString()

Get the trigger soure as a string value for the current channel.

Prototype	*Unicode* `JSR_Status GetPulserVoltsString(BSTR pString);`
	Ascii `JSR_Status GetPulserVoltsString(char* pString);`
Base	`GetPropertyDisplayString(JSR_ID_PulserVolts, pString);`

Call	*Unicode*
	`JSR_String jsrString;`
	`JSR_Status status = ic.GetPulserVoltsString(jsrString);`

	Ascii
	`JSR_Ascii asciiString;`
	`JSR_Status status = ic.GetPulserVoltsString(asciiString);`

Return	JSR_OK if successful. The pulser PRF string value is set to parameter string value.

6.8.7 JSR_ID_PulserDampResistorIndex

6.8.7.1 GetPulserDampResistorIndex()

Get the pulser damp resistor index for the current channel.

Prototype	`JSR_Status GetPulserDampResistorIndex(JSR_Int32* pIndex);`
Base	`GetInt32(JSR_ID_PulserDampResistorIndex, pIndex);`

Call	`JSR_Int32 nIndex;`
	`JSR_Status status = ic.GetPulserDampResistorIndex(&nIndex);`

Return	JSR_OK if successful. The pulser damp resistor index value is return by parameter reference.

6.8.7.2 SetPulserDampResistorIndex()

Set the pulser damp resistor index for the current channel.

Prototype	`JSR_Status SetPulserDampResistorIndex(JSR_Int32 nIndex);`
Base	`SetInt32(JSR_ID_PulserDampResistorIndex, nIndex);`
Call	`JSR_Status status = ic.SetPulserDampResistorIndex(1);`
Return	JSR_OK if successful.

6.8.7.3 GetPulserDampResistorIndexMin()

Get the minimum pulser damp resistor index value for the current channel.

Prototype	`JSR_Status GetPulserDampResistorIndexMin(JSR_Int32* pIndex);`
Base	`GetPropertyMinValueInt32(JSR_ID_PulserDampResistorIndex, pIndex);`

Call	`JSR_Int32 nIndex;`
	`JSR_Status status = ic.GetPulserDampResistorIndexMin(&nIndex);`

Return	JSR_OK if successful. The minimum pulser damp resistor index value is return by parameter reference.

6.8.7.4 GetPulserDampResistorIndexMax()

Get the maximum pulser damp resistor index value for the current channel.

Prototype	`JSR_Status GetPulserDampResistorIndexMax(JSR_Int32* pIndex);`
Base	`GetPropertyMaxValueInt32(JSR_ID_PulserDampResistorIndex, pIndex);`

Call	`JSR_Int32 nIndex;`
	`JSR_Status status = ic.GetPulserDampResistorIndexMax(&nIndex);`

Return	JSR_OK if successful. The maximum pulser damp resistor index value is return by parameter reference.

6.8.7.5 GetPulserDampResistorOhms()

Get the pulser damp resistor in Ohms for the current channel.

Prototype	`JSR_Status GetPulserDampResistorOhms(JSR_Double* pVal);`
Base	`GetIndexValueDouble(JSR_ID_PulserDampResistorIndex, pVal);`

Call	`JSR_Double fOhms;`
	`JSR_Status status = ic.GetPulserDampResistorOhms(&fOhms);`

Return	JSR_OK if successful. The pulser damp resistor in Ohms is return by parameter reference.

6.8.7.6 GetPulserDampResistorOhmsString()

Get the pulser damp resistor as a string value for the current channel.

Prototype	*Unicode* `JSR_Status GetPulserDampResistorOhmsString(BSTR pVal);`
	Ascii `JSR_Status GetPulserDampResistorOhmsString(char* pVal);`
Base	`GetPropertyDisplayString(JSR_ID_PulserDampResistorIndex, pVal);`

Call	*Unicode*
	`JSR_String jsrString;`
	`JSR_Status status = ic.GetPulserDampResistorOhmsString(jsrString);`

	Ascii
	`JSR_Ascii asciiString;`
	`JSR_Status status = ic.GetPulserDampResistorOhmsString(asciiString);`

Return	JSR_OK if successful. The pulser damp resistor string value is set to parameter string value.

6.8.8 JSR_ID_PulserEnergyIndex

6.8.8.1 GetPulserEnergyIndex()

Get the pulser energy index for the current channel.

Prototype	`JSR_Status GetPulserEnergyIndex(JSR_Int32* pIndex);`
Base	`GetInt32(JSR_ID_PulserEnergyIndex, pIndex);`

Call	`JSR_Int32 nIndex;`
	`JSR_Status status = ic.GetPulserEnergyIndex(&nIndex);`

Return	JSR_OK if successful. The pulser energy index value is return by parameter reference.

6.8.8.2 SetPulserEnergyIndex()

Set the pulser energy index for the current channel.

Prototype	`JSR_Status SetPulserEnergyIndex(JSR_Int32 nIndex);`
Base	`SetInt32(JSR_ID_PulserEnergyIndex, nIndex);`
Call	`JSR_Status status = ic.SetPulserEnergyIndex(1);`
Return	JSR_OK if successful.

6.8.8.3 GetPulserEnergyIndexMin()

Get the minimum pulser energy index value for the current channel.

Prototype	`JSR_Status GetPulserEnergyIndexMin(JSR_Int32* pIndex);`
Base	`GetPropertyMinValueInt32(JSR_ID_PulserEnergyIndex, pIndex);`

Call	`JSR_Int32 nIndex;`
	`JSR_Status status = ic.GetPulserEnergyIndexMin(&nIndex);`

Return	JSR_OK if successful. The minimum pulser energy index value is return by parameter reference.

6.8.8.4 GetPulserEnergyIndexMax()

Get the minimum pulser energy index value for the current channel.

Prototype	`JSR_Status GetPulserEnergyIndexMin(JSR_Int32* pIndex);`
Base	`GetPropertyMaxValueInt32(JSR_ID_PulserEnergyIndex, pIndex);`

Call	`JSR_Int32 nIndex;`
	`JSR_Status status = ic.GetPulserEnergyIndexMax(&nIndex);`

Return	JSR_OK if successful. The maximum pulser energy index value is return by parameter reference.

6.8.8.5 GetPulserEnergy()

Get the pulser energy value for the current channel.

Prototype	`JSR_Status GetPulserEnergy(JSR_Double* pVal);`
Base	`GetIndexValueDouble(JSR_ID_PulserEnergyIndex, pVal);`

Call	`JSR_Double fEnergy;`
	`JSR_Status status = ic.GetPulserEnergy(&fEnergy);`

Return	JSR_OK if successful. The pulser energy value is return by parameter reference.

6.8.8.6 GetPulserEnergyString()

Get the pulser energy as a string value for the current channel.

Prototype	*Unicode* `JSR_Status GetPulserEnergyString(BSTR pString);`
	Ascii `JSR_Status GetPulserEnergyString(char* pString);`
Base	`GetPropertyDisplayString(JSR_ID_PulserEnergyIndex, pString);`

Call	*Unicode*
	`JSR_String jsrString;`
	`JSR_Status status = ic.GetPulserEnergyString(jsrString);`

	Ascii
	`JSR_Ascii asciiString;`
	`JSR_Status status = ic.GetPulserEnergyString(asciiString);`

Return	JSR_OK if successful. The pulser energy string value is set to parameter string value.

6.8.9 JSR_ID_PulserEnergyPerPulse

6.8.9.1 GetPulserEnergyPerPulse()

Get the pulser energy per pulse value for the current channel.

Prototype	`JSR_Status GetPulserEnergyPerPulse(JSR_Double* pVal);`
Base	`GetDouble(JSR_ID_PulserEnergyPerPulse, pVal);`

Call	`JSR_Double fEnergyPerPulse;`
	`JSR_Status status = ic.GetPulserEnergy(&fEnergyPerPulse);`

Return	JSR_OK if successful. The pulser energy value is return by parameter reference.

6.8.9.2 GetPulserEnergyPerPulseString()

Get the pulser energy per pulse as a string value for the current channel.

Prototype	*Unicode* `JSR_Status GetPulserEnergyPerPulseString(BSTR pString);`
	Ascii `JSR_Status GetPulserEnergyPerPulseString(char* pString);`
Base	`GetPropertyDisplayString(JSR_ID_PulserEnergyPerPulse, pString);`

Call	*Unicode*
	`JSR_String jsrString;`
	`JSR_Status status = ic.GetPulserEnergyPerPulseString(jsrString);`

	Ascii
	`JSR_Ascii asciiString;`
	`JSR_Status status = ic.GetPulserEnergyPerPulseString(asciiString);`

Return	JSR_OK if successful. The pulser energy per pulse string value is set to parameter string value.

6.8.10 JSR_ID_PulserLEDControl

6.8.10.1 GetPulserLEDControl()

Get the pulser LED control value for the current channel.

Prototype	`JSR_Status GetPulserLEDControl(JSR_PulserBlinkEnum* pVal);`
Base	`GetInt32(JSR_ID_PulserLEDControl, (JSR_Int32*)pVal);`

Call	`JSR_PulserBlinkEnum nVal;`
	`JSR_Status status = ic.GetPulserLEDControl(&nVal);`

Return	JSR_OK if successful. The pulser LED control value is return by parameter reference.

6.8.10.2 SetPulserLEDControl()

Set the pulser LED control value for the current channel.

Prototype	`JSR_Status SetPulserLEDControl(JSR_PulserBlinkEnum eVal);`
Base	`SetInt32(JSR_ID_PulserLEDControl, (JSR_Int32)eVal);`
Call	`JSR_Status status = ic.SetPulserLEDControl(JSR_LED_IDENTIFY_BOARD);`
Return	JSR_OK if successful.

6.8.10.3 GetPulserLEDControlString()

Get the pulser LED control value as a string value for the current channel.

Prototype	*Unicode* `JSR_Status GetPulserLEDControlString(BSTR pString);`
	Ascii `JSR_Status GetPulserLEDControlString(char* pString);`
Base	`GetPropertyDisplayString(JSR_ID_PulserLEDControl, pString);`

Call	*Unicode*
	`JSR_String jsrString;`
	`JSR_Status status = ic.GetPulserLEDControlString(jsrString);`

	Ascii
	`JSR_Ascii asciiString;`
	`JSR_Status status = ic.GetPulserLEDControlString(asciiString);`

Return	JSR_OK if successful. The pulser LED control string value is set to parameter string value.

6.8.11 JSR_ID_PulserIsPulsing

6.8.11.1 GetPulserIsPulsing()

Get the pulser LED control value for the current channel.

Prototype	`JSR_Status GetPulserIsPulsing(JSR_BoolEnum* pBool);`
Base	`GetBool(JSR_ID_PulserIsPulsing, pBool);`

Call	`JSR_BoolEnum bVal;`
	`JSR_Status status = ic.GetPulserIsPulsing(&bVal);`

Return	JSR_OK if successful. The pulser is pulsing value is return by parameter reference.

6.8.11.2 GetPulserIsPulsingString()

Get the pulser is pulsing value as a string value for the current channel.

Prototype	*Unicode* `JSR_Status GetPulserIsPulsingString(BSTR pString);`
	Ascii `JSR_Status GetPulserIsPulsingString(char* pString);`
Base	`etPropertyDisplayString(JSR_ID_PulserIsPulsing, pString);`

Call	*Unicode*
	`JSR_String jsrString;`
	`JSR_Status status = ic.GetPulserIsPulsingString(jsrString);`

	Ascii
	`JSR_Ascii asciiString;`
	`JSR_Status status = ic.GetPulserIsPulsingString(asciiString);`

Return	JSR_OK if successful. The pulser is pulsing string value is set to parameter string value.

6.8.12 JSR_ID_PulserPowerLimitStatus

6.8.12.1 GetPulserPowerLimitStatus()

Get the pulser power limit status for the current channel.

Prototype	`JSR_Status GetPulserPowerLimitStatus(JSR_Status* pStatus);`
Base	`GetInt32(JSR_ID_PulserPowerLimitStatus, (JSR_Int32*)pStatus);`

Call	`JSR_Status nStatus;`
	`JSR_Status status = ic.GetPulserPowerLimitStatus(&nStatus);`

Return	JSR_OK if successful. The pulser power limit status is return by parameter reference.

6.8.12.2 GetPulserPowerLimitStatusString()

Get the pulser power limit status as a string value for the current channel.

Prototype	*Unicode* `JSR_Status GetPulserPowerLimitStatusString(BSTR pString);`
	Ascii `JSR_Status GetPulserPowerLimitStatusString(char* pString);`
Base	`GetPropertyDisplayString(JSR_ID_PulserPowerLimitStatus, pString);`

Call	*Unicode*
	`JSR_String jsrString;`
	`JSR_Status status = ic.GetPulserPowerLimitStatusString(jsrString);`

	Ascii
	`JSR_Ascii asciiString;`
	`JSR_Status status = ic.GetPulserPowerLimitStatusString(asciiString);`

Return	JSR_OK if successful. The pulser power limit status string is set to parameter string value.

6.8.13 JSR_ID_PulserPowerLimitPRF

6.8.13.1 GetPulserPowerLimitPRF()

Get the pulser power limit PRF value for the current channel.

Prototype	`JSR_Status GetPulserPowerLimitPRF(JSR_Double* pVal);`
Base	`GetDouble(JSR_ID_PulserPowerLimitPRF, pVal);`

Call	`JSR_Double fPRF;`
	`JSR_Status status = ic.GetPulserPowerLimitPRF(&fPRF);`

Return	JSR_OK if successful. The pulser power limit PRF value is return by parameter reference.

6.8.13.2 GetPulserPowerLimitPRFString()

Get the pulser power limit PRF value as a string value for the current channel.

Prototype	*Unicode* `JSR_Status GetPulserPowerLimitPRFString(BSTR pString);`
	Ascii `JSR_Status GetPulserPowerLimitPRFString(char* pString);`
Base	`GetPropertyDisplayString(JSR_ID_PulserPowerLimitPRF, pString);`

Call	*Unicode*
	`JSR_String jsrString;`
	`JSR_Status status = ic.GetPulserPowerLimitPRFString(jsrString);`

	Ascii
	`JSR_Ascii asciiString;`
	`JSR_Status status = ic.GetPulserPowerLimitPRFString(asciiString);`

Return	JSR_OK if successful. The pulser power limit PRF string is set to parameter string value.

6.8.14 JSR_ID_PulserPowerLimitVolts

6.8.14.1 GetPulserPowerLimitVolts()

Get the pulser power limit volts value for the current channel.

Prototype	`JSR_Status GetPulserPowerLimitVolts(JSR_Double* pVal);`
Base	`GetDouble(JSR_ID_PulserPowerLimitVolts, pVal);`

Call	`JSR_Double fVolts;`
	`JSR_Status status = ic.GetPulserPowerLimitVolts(&fVolts);`

Return	JSR_OK if successful. The pulser power limit volts value is return by parameter reference.

6.8.14.2 GetPulserPowerLimitVoltsString()

Get the pulser power limit volts value as a string value for the current channel.

Prototype	*Unicode* `JSR_Status GetPulserPowerLimitVoltsString(BSTR pString);`
	Ascii `JSR_Status GetPulserPowerLimitVoltsString(char* pString);`
Base	`GetPropertyDisplayString(JSR_ID_PulserPowerLimitVolts, pString);`

Call	*Unicode*
	`JSR_String jsrString;`
	`JSR_Status status = ic.GetPulserPowerLimitVoltsString(jsrString);`

	Ascii
	`JSR_Ascii asciiString;`
	`JSR_Status status = ic.GetPulserPowerLimitVoltsString(asciiString);`

Return	JSR_OK if successful. The pulser power limit volts string is set to parameter string value.

6.8.15 JSR_ID_PulserPowerLimitEnergyIndex

6.8.15.1 GetPulserPowerLimitEnergyIndex()

Get the pulser power limit energy index for the current channel.

Prototype	`JSR_Status GetPulserPowerLimitEnergyIndex(JSR_Int32* pIndex);`
Base	`GetInt32(JSR_ID_PulserPowerLimitEnergyIndex, pIndex);`

Call	`JSR_Int32 nIndex;`
	`JSR_Status status = ic.GetPulserPowerLimitEnergyIndex(&nIndex);`

Return	JSR_OK if successful. The pulser power limit energy index value is return by parameter reference.

6.8.15.2 GetPulserPowerLimitEnergy()

Get the pulser power limit energy value for the current channel.

Prototype	`JSR_Status GetPulserPowerLimitEnergy(JSR_Double* pVal);`
Base	`GetIndexValueDouble(JSR_ID_PulserPowerLimitEnergyIndex, pVal);`

Call	`JSR_Double fEnergy;`
	`JSR_Status status = ic.GetPulserPowerLimitEnergyIndex(&fEnergy);`

Return	JSR_OK if successful. The pulser power limit energy value is return by parameter reference.

6.8.15.3 GetPulserPowerLimitEnergyString()

Get the pulser power limit energy value as a string value for the current channel.

Prototype	*Unicode* `JSR_Status GetPulserPowerLimitEnergyString(BSTR pString)`
	Ascii `JSR_Status GetPulserPowerLimitEnergyString(char* pString);`
Base	`GetPropertyDisplayString(JSR_ID_PulserPowerLimitEnergyIndex, pString);`

Call	*Unicode*
	`JSR_String jsrString;`
	`JSR_Status status = ic.GetPulserPowerLimitEnergyString(jsrString);`

	Ascii
	`JSR_Ascii asciiString;`
	`JSR_Status status = ic.GetPulserPowerLimitEnergyString(asciiString);`

Return	JSR_OK if successful. The pulser power limit energy value string is set to parameter string value.

6.8.16 JSR_ID_ReceiverBandwidth

6.8.16.1 GetReceiverBandwidth()

Get the receiver bandwidth value for the current channel.

Prototype	`JSR_Status GetReceiverBandwidth(JSR_Int32* pVal);`
Base	`GetInt32(JSR_ID_ReceiverBandwidth, pVal);`

Call	`JSR_Int32 bwMHz;`
	`JSR_Status status = ic.GetReceiverBandwidth(&bwMHz);`

Return	JSR_OK if successful. The receiver bandwidth value is return by parameter reference.

6.8.16.2 GetReceiverBandwidthString()

Get the receiver bandwidth value as a string value for the current channel.

Prototype	*Unicode* `JSR_Status GetReceiverBandwidthString(BSTR pString);`
	Ascii `JSR_Status GetReceiverBandwidthString(char* pString);`
Base	`GetPropertyDisplayString(JSR_ID_ReceiverBandwidth, pString);`

Call	*Unicode*
	`JSR_String jsrString;`
	`JSR_Status status = ic.GetReceiverBandwidthString(jsrString);`

	Ascii
	`JSR_Ascii asciiString;`
	`JSR_Status status = ic.GetReceiverBandwidthString(asciiString);`

Return	JSR_OK if successful. The receiver bandwidth value string is set to parameter string value.

6.8.17 JSR_ID_ReceiverSignalSelect

6.8.17.1 GetReceiverSignalSelect()

Get the receiver signal select value for the current channel.

Prototype	`JSR_Status GetReceiverSignalSelect(JSR_SignalSelectEnum* pVal);`
Base	`GetInt32(JSR_ID_ReceiverSignalSelect, (JSR_Int32*)pVal);`

Call	`JSR_SignalSelectEnum nSelect;`
	`JSR_Status status = ic.GetReceiverSignalSelect(&nSelect);`

Return	JSR_OK if successful. The receiver signal select value is return by parameter reference.

6.8.17.2 SetReceiverSignalSelect()

Set the receiver signal select value for the current channel.

Prototype	`JSR_Status SetReceiverSignalSelect(JSR_SignalSelectEnum eVal);`
Base	`SetInt32(JSR_ID_ReceiverSignalSelect, (JSR_Int32)eVal);`
Call	`JSR_Status status = ic.GetReceiverSignalSelect(JSR_SIGNAL_SELECT_BOTH);`
Return	JSR_OK if successful.

6.8.17.3 GetReceiverSignalSelectString()

Get the receiver signal select value as a string value for the current channel.

Prototype	*Unicode* `JSR_Status GetReceiverSignalSelectString(BSTR pString);`
	Ascii `JSR_Status GetReceiverSignalSelectString(char* pString);`
Base	`GetPropertyDisplayString(JSR_ID_ReceiverSignalSelect, pString);`

Call	*Unicode*
	`JSR_String jsrString;`
	`JSR_Status status = ic.GetReceiverSignalSelectString(jsrString);`

	Ascii
	`JSR_Ascii asciiString;`
	`JSR_Status status = ic.GetReceiverSignalSelectString(asciiString);`

Return	JSR_OK if successful. The receiver signal select value string is set to parameter string value.

6.8.18 JSR_ID_ReceiverGainDB

6.8.18.1 GetReceiverGainDB()

Get the receiver gain DB value for the current channel.

Prototype	`JSR_Status GetReceiverGainDB(JSR_Double* pVal);`
Base	`GetDouble(JSR_ID_ReceiverGainDB, pVal);`

Call	`JSR_Double fGain;`
	`JSR_Status status = ic.GetReceiverGainDB(&fGain);`

Return	JSR_OK if successful. The receiver gain DB value is return by parameter reference.

6.8.18.2 SetReceiverGainDB()

Set the receiver gain DB value for the current channel.

Prototype	`JSR_Status SetReceiverGainDB(JSR_Double fVal);`
Base	`SetDouble(JSR_ID_ReceiverGainDB, fVal);`
Call	`JSR_Status status = ic.SetReceiverGainDB(10.0);`
Return	JSR_OK if successful.

6.8.18.3 GetReceiverGainDBMin()

Get the minimum receiver gain DB value for the current channel.

Prototype	`JSR_Status GetReceiverGainDBMin(JSR_Double* pVal);`
Base	`GetPropertyMinValueDouble(JSR_ID_ReceiverGainDB, pVal);`

Call	`JSR_Double fGain;`
	`JSR_Status status = ic.GetReceiverGainDBMin(&fGain);`

Return	JSR_OK if successful. The minimum receiver gain DB value is return by parameter reference.

6.8.18.4 GetReceiverGainDBMax()

Get the maximum receiver gain DB value for the current channel.

Prototype	`JSR_Status GetReceiverGainDBMax(JSR_Double* pVal);`
Base	`GetPropertyMaxValueDouble(JSR_ID_ReceiverGainDB, pVal);`

Call	`JSR_Double fGain;`
	`JSR_Status status = ic.GetReceiverGainDBMax(&fGain);`

Return	JSR_OK if successful. The maximum receiver gain DB value is return by parameter reference.

6.8.18.5 GetReceiverGainDBString()

Get the receiver gain DB value as a string value for the current channel.

Prototype	*Unicode* `JSR_Status GetReceiverGainDBString(BSTR pString);`
	Ascii `JSR_Status GetReceiverGainDBString(char* pString);`
Base	`GetPropertyDisplayString(JSR_ID_ReceiverGainDB, pString);`

Call	*Unicode*
	`JSR_String jsrString;`
	`JSR_Status status = ic.GetReceiverGainDBString(jsrString);`

	Ascii
	`JSR_Ascii asciiString;`
	`JSR_Status status = ic.GetReceiverGainDBString(asciiString);`

Return	JSR_OK if successful. The receiver gain DB value string is set to parameter string value.

6.8.19 JSR_ID_ReceiverThroughGainDB

6.8.19.1 GetReceiverThroughGainDB()

Get the receiver through gain DB value for the current channel.

Only the PRC50 supports this property.

Prototype	`JSR_Status GetReceiverThroughGainDB(JSR_Double* pVal);`
Base	`GetDouble(JSR_ID_ReceiverThroughGainDB, pVal);`

Call	`JSR_Double fGain;`
	`JSR_Status status = ic.GetReceiverThroughGainDB(&fGain);`

Return	JSR_OK if successful. The receiver through gain DB value is return by parameter reference.

6.8.19.2 SetReceiverThroughGainDB()

Set the receiver through gain DB value for the current channel.

Only the PRC50 supports this property.

Prototype	`JSR_Status SetReceiverThroughGainDB(JSR_Double fVal);`
Base	`SetDouble(JSR_ID_ReceiverThroughGainDB, fVal);`
Call	`JSR_Status status = ic.SetReceiverThroughGainDB(10.0);`
Return	JSR_OK if successful.

6.8.19.3 GetReceiverThroughGainDBMin()

Get the minimum receiver through gain DB value for the current channel.

Only the PRC50 supports this property.

Prototype	`JSR_Status GetReceiverThroughGainDBMin(JSR_Double* pVal);`
Base	`GetPropertyMinValueDouble(JSR_ID_ReceiverThroughGainDB, pVal);`

Call	`JSR_Double fGain;`
	`JSR_Status status = ic.GetReceiverThroughGainDBMin(&fGain);`

Return	JSR_OK if successful. The minimum receiver through gain DB value is return by parameter reference.

6.8.19.4 GetReceiverThroughGainDBMax()

Get the maximum receiver through gain DB value for the current channel.

Only the PRC50 supports this property.

Prototype	`JSR_Status GetReceiverThroughGainDBMax(JSR_Double* pVal);`
Base	`GetPropertyMaxValueDouble(JSR_ID_ReceiverThroughGainDB, pVal);`

Call	`JSR_Double fGain;`
	`JSR_Status status = ic.GetReceiverThroughGainDBMax(&fGain);`

Return	JSR_OK if successful. The maximum receiver through gain DB value is return by parameter reference.

6.8.20 JSR_ID_ReceiverTREchoGainDB

6.8.20.1 GetReceiverTREchoGainDB()

Get the receiver TR Echo gain DB value for the current channel.

Only the PRC50 supports this property.

Prototype	`JSR_Status GetReceiverTREchoGainDB(JSR_Double* pVal);`
Base	`GetDouble(JSR_ID_ReceiverTREchoGainDB, pVal);`

Call	`JSR_Double fGain;`
	`JSR_Status status = ic.GetReceiverTREchoGainDB(&fGain);`

Return	JSR_OK if successful. The receiver TR Echo gain DB value is return by parameter reference.

6.8.20.2 SetReceiverTREchoGainDB()

Set the receiver TR Echo gain DB value for the current channel.

Only the PRC50 supports this property.

Prototype	`JSR_Status SetReceiverTREchoGainDB(JSR_Double fVal);`
Base	`SetDouble(JSR_ID_ReceiverTREchoGainDB, fVal);`
Call	`JSR_Status status = ic.SetReceiverTREchoGainDB(10.0);`
Return	JSR_OK if successful.

6.8.20.3 GetReceiverTREchoGainDBMin()

Get the minimum receiver TR Echo gain DB value for the current channel.

Only the PRC50 supports this property.

Prototype	`JSR_Status GetReceiverTREchoGainDBMin(JSR_Double* pVal);`
Base	`GetPropertyMinValueDouble(JSR_ID_ReceiverTREchoGainDB, pVal);`

Call	`JSR_Double fGain;`
	`JSR_Status status = ic.GetReceiverTREchoGainDBMin(&fGain);`

Return	JSR_OK if successful. The minimum receiver TR Echo gain DB value is return by parameter reference.

6.8.20.4 GetReceiverTREchoGainDBMax()

Get the maximum receiver TR Echo gain DB value for the current channel.

Only the PRC50 supports this property.

Prototype	`JSR_Status GetReceiverTREchoGainDBMax(JSR_Double* pVal);`
Base	`GetPropertyMaxValueDouble(JSR_ID_ReceiverTREchoGainDB, pVal);`

Call	`JSR_Double fGain;`
	`JSR_Status status = ic.GetReceiverTREchoGainDBMax(&fGain);`

Return	JSR_OK if successful. The maximum receiver TR Echo gain DB value is return by parameter reference.

6.8.20.5 GetReceiverTREchoGainDBString()

Get the receiver TR Echo gain DB value as a string value for the current channel.

Only the PRC50 supports this property.

Prototype	*Unicode* `JSR_Status GetReceiverTREchoGainDBString(BSTR pString);`
	Ascii `JSR_Status GetReceiverTREchoGainDBString(char* pString);`
Base	`GetPropertyDisplayString(JSR_ID_ReceiverTREchoGainDB, pString);`

Call	*Unicode*
	`JSR_String jsrString;`
	`JSR_Status status = ic.GetReceiverTREchoGainDBString(jsrString);`

	Ascii
	`JSR_Ascii asciiString;`
	`JSR_Status status = ic.GetReceiverTREchoGainDBString(asciiString);`

Return	JSR_OK if successful. The receiver TR Echo gain DB value string is set to parameter string value.

6.8.21 JSR_ID_ReceiverLPFilterIndex

6.8.21.1 GetReceiverLPFilterIndex()

Get the receiver low pass filter index value for the current channel.

Prototype	`JSR_Status GetReceiverLPFilterIndex(JSR_Int32* pIndex);`
Base	`GetInt32(JSR_ID_ReceiverLPFilterIndex, pIndex);`

Call	`JSR_Int32 nIndex;`
	`JSR_Status status = ic.GetReceiverLPFilterIndex(&nIndex);`

Return	JSR_OK if successful. The receiver low pass filter index is return by parameter reference.

6.8.21.2 SetReceiverLPFilterIndex()

Set the receiver low pass filter index value for the current channel.

Prototype	`JSR_Status SetReceiverLPFilterIndex(JSR_Int32 nIndex);`
Base	`SetInt32(JSR_ID_ReceiverLPFilterIndex, nIndex);`
Call	`JSR_Status status = ic.SetReceiverLPFilterIndex(1);`
Return	JSR_OK if successful.

6.8.21.3 GetReceiverLPFilterIndexMin()

Get the minimum receiver low pass filter index value for the current channel.

Prototype	`JSR_Status GetReceiverLPFilterIndexMin(JSR_Int32* pIndex);`
Base	`GetPropertyMinValueInt32(JSR_ID_ReceiverLPFilterIndex, pIndex);`

Call	`JSR_Int32 nIndex;`
	`JSR_Status status = ic.GetReceiverLPFilterIndexMin(&nIndex);`

Return	JSR_OK if successful. The minimum receiver low pass filter index is return by parameter reference.

6.8.21.4 GetReceiverLPFilterIndexMax()

Get the maximum receiver low pass filter index value for the current channel.

Prototype	`JSR_Status GetReceiverLPFilterIndexMax(JSR_Int32* pIndex);`
Base	`GetPropertyMaxValueInt32(JSR_ID_ReceiverLPFilterIndex, pIndex);`

Call	`JSR_Int32 nIndex;`
	`JSR_Status status = ic.GetReceiverLPFilterIndexMax(&nIndex);`

Return	JSR_OK if successful. The maximum receiver low pass filter index is return by parameter reference.

6.8.21.5 GetReceiverLPFilterMHz()

Get the receiver low pass filter value for the current channel.

Prototype	`JSR_Status GetReceiverLPFilterMHz(JSR_Double* pVal);`
Base	`GetIndexValueDouble(JSR_ID_ReceiverLPFilterIndex, pVal);`

Call	`JSR_Double fMHz;`
	`JSR_Status status = ic.GetReceiverLPFilterMHz(&fMHz);`

Return	JSR_OK if successful. The receiver low pass filter value is return by parameter reference.

6.8.21.6 GetReceiverLPFilterMHzString()

Get the receiver low pass filter value as a string value for the current channel.

Prototype	*Unicode* `JSR_Status GetReceiverLPFilterMHzString(BSTR pString);`
	Ascii `JSR_Status GetReceiverLPFilterMHzString(char* pString);`
Base	`GetPropertyDisplayString(JSR_ID_ReceiverLPFilterIndex, pString);`

Call	*Unicode*
	`JSR_String jsrString;`
	`JSR_Status status = ic.GetReceiverLPFilterMHzString(jsrString);`

	Ascii
	`JSR_Ascii asciiString;`
	`JSR_Status status = ic.GetReceiverLPFilterMHzString(asciiString);`

Return	JSR_OK if successful. The receiver low pass filter value string is set to parameter string value.

6.8.22 JSR_ID_ReceiverHPFilterIndex

6.8.22.1 GetReceiverHPFilterIndex()

Get the receiver high pass filter index value for the current channel.

Prototype	`JSR_Status GetReceiverHPFilterIndex(JSR_Int32* pIndex);`
Base	`GetInt32(JSR_ID_ReceiverHPFilterIndex, pIndex);`

Call	`JSR_Int32 nIndex;`
	`JSR_Status status = ic.GetReceiverHPFilterIndex(&nIndex);`

Return	JSR_OK if successful. The receiver high pass filter index is return by parameter reference.

6.8.22.2 SetReceiverHPFilterIndex()

Set the receiver high pass filter index value for the current channel.

Prototype	`JSR_Status SetReceiverHPFilterIndex(JSR_Int32 nIndex);`
Base	`SetInt32(JSR_ID_ReceiverHPFilterIndex, nIndex);`
Call	`JSR_Status status = ic.SetReceiverHPFilterIndex(1);`
Return	JSR_OK if successful.

6.8.22.3 GetReceiverHPFilterIndexMin()

Get the minimum receiver high pass filter index value for the current channel.

Prototype	`JSR_Status GetReceiverHPFilterIndexMin(JSR_Int32* pIndex);`
Base	`GetPropertyMinValueInt32(JSR_ID_ReceiverHPFilterIndex, pIndex);`

Call	`JSR_Int32 nIndex;`
	`JSR_Status status = ic.GetReceiverHPFilterIndexMin(&nIndex);`

Return	JSR_OK if successful. The minimum receiver high pass filter index is return by parameter reference.

6.8.22.4 GetReceiverHPFilterIndexMax()

Get the maximum receiver high pass filter index value for the current channel.

Prototype	`JSR_Status GetReceiverHPFilterIndexMax(JSR_Int32* pIndex);`
Base	`GetPropertyMaxValueInt32(JSR_ID_ReceiverHPFilterIndex, pIndex);`

Call	`JSR_Int32 nIndex;`
	`JSR_Status status = ic.GetReceiverHPFilterIndexMax(&nIndex);`

Return	JSR_OK if successful. The maximum receiver high pass filter index is return by parameter reference.

6.8.22.5 GetReceiverHPFilterMHz()

Get the receiver high pass filter value for the current channel.

Prototype	`JSR_Status GetReceiverHPFilterMHz(JSR_Double* pVal);`
Base	`GetIndexValueDouble(JSR_ID_ReceiverHPFilterIndex, pVal);`

Call	`JSR_Double fMHz;`
	`JSR_Status status = ic.GetReceiverHPFilterMHz(&fMHz);`

Return	JSR_OK if successful. The receiver high pass filter value is return by parameter reference.

6.8.22.6 GetReceiverHPFilterMHzString()

Get the receiver high pass filter value as a string value for the current channel.

Prototype	*Unicode* `JSR_Status GetReceiverHPFilterMHzString(BSTR pString);`
	Ascii `JSR_Status GetReceiverHPFilterMHzString(char* pString);`
Base	`GetPropertyDisplayString(JSR_ID_ReceiverHPFilterIndex, pString);`

Call	*Unicode*
	`JSR_String jsrString;`
	`JSR_Status status = ic.GetReceiverHPFilterMHzString(jsrString);`

	Ascii
	`JSR_Ascii asciiString;`
	`JSR_Status status = ic.GetReceiverHPFilterMHzString(asciiString);`

Return	JSR_OK if successful. The receiver high pass filter value string is set to parameter string value.

7 JSR_Simple_ActiveX Library

7.1 Declare JSR_Simple

To access the JSR_Simple_ActiveX library in Visual Basic, you must declare a variable to access the JSR_Simple interface.

Example:

```
Dim ic as New JSR_Simple
```

All the Call, Get and Put examples in this section use “ic” to access the function or property.

7.2 Library Functions

7.2.1 Open()

Open JSR_Common Library. Check LoadLibraryStatus and OpenLibraryStatus to determin status of Open().

Parameters	`Open(JSR_LibOpenOptionsEnum, JSR_ModelEnum)`
Call	`Call ic.Open(JSR_LIB_OPTION_DEFAULT, JSR_MODEL_ANY)`

7.2.2 Close

Close the JSR_Common Library.

Parameters	None
Call	`Call ic.Close`

7.3 Library Properties

7.3.1 LoadLibraryStatus

Status of loading JSR_Common. JSR_OK if no errors.

Property Type	JSR_Status
Parameters	None

Get	`Dim status as JSR_Status`
	`status = ic.LoadLibraryStatus`

Put	Property is Read Only.

7.3.2 OpenLibraryStatus

Status of opening JSR_Common. JSR_OK if no errors.

Property Type	JSR_Status
Parameters	None

Get	`Dim status as JSR_Status`
	`status = ic.OpenLibraryStatus`

Put	Property is Read Only.

7.3.3 IsOpen

State of JSR_Common Library. *True* if open else *False.*

Property Type	Bool
Parameters	None

Get	`Dim bVal as Bool`
	`bVal = ic.IsOpen`

Put	Property is Read Only.

7.3.4 SupportedDrivers

String containing list of supported drivers.

Property Type	String
Parameters	None

Get	`Dim str as String`
	`str = ic.SupportedDrivers`

Put	Property is Read Only.

7.3.5 IsDriverSupported

String containing list of supported drivers.

Property Type	Bool
Parameters	None

Get	`Dim bVal as Bool`
	`bVal = ic.SupportedDrivers`

Put	Property is Read Only.

7.3.6 IsModelSupported

Determine if the JSR_Common library supports model.

Property Type	Bool
Parameters	`IsModelSupported(JSR_ModelEnum)`

Get	`Dim bVal as Bool`
	`bVal = ic.IsModelSupported(JSR_MODEL_PRC50)`

Put	Property is Read Only.

7.3.7 DataTypeName

String containing the name of the data type supplied by the JSR_TypeEnum parameter.

Property Type	String
Parameters	`DataTypeName(JSR_TypeEnum)`

Get	`Dim name as String`
	`name = ic.DataTypeName(JSR_TYPE_ENUM_STRING)`

Put	Property is Read Only.

7.4 Status and Diagnostic Properties

7.4.1 LastError

Status of last property operation.

Property Type	`JSR_Status`
Parameters	None

Get	`Dim status as JSR_Status`
	`status = ic.LastError`

Put	Property is Read Only.

7.4.2 LastErrorString

Status string of last property operation.

Property Type	String
Parameters	None

Get	`Dim errstr as String`
	`errstr = ic.LastErrorString`

Put	Property is Read Only.

7.4.3 ErrorString

Status string of supplied JSR_Status parameter.

Property Type	String
Parameters	`ErrorString(JSR_Status)`

Get	`Dim errstr as String`
	`errstr = ic.ErrorString(JSR_OK)`

Put	Property is Read Only.

7.4.4 LastProperty

Property Id that was operated. Used for error handling and diagnostics.

Property Type	`JSR_PropertyIDEnum`
Parameters	None

Get	`Dim propid as JSR_PropertyIDEnum`
	`propid = ic.LastProperty`

Put	Property is Read Only.

7.5 Channel Properties

7.5.1 ChannelCount

Number of Channels available by the opened JSR_Simple library.

Property Type	Integer
Parameters	None

Get	`Dim num as Integer`
	`num = ic.ChannelCount`

Put	Property is Read Only.

7.5.2 Channel

Current channel for property operations. 0 to ChannelCount – 1.

Property Type	Integer
Parameters	None

Get	`Dim channel as Integer`
	`channel = ic.Channel`

Put	Ic.Channel = 1

7.5.3 ChannelDescription

String containing the channel description.

Same as the value for JSR_Common property `JSR_ID_ChannelDescription`.

Property Type	String
Parameters	None

Get	`Dim desc as String`
	`desc = ic.ChannelDescription`

Put	Property is Read Only.

7.5.4 ChannelLetter

String containing the channel description.

Same as the value for JSR_Common property `JSR_ID_ChannelLetter.`

Property Type	String
Parameters	None

Get	`Dim chanletter as String`
	`chanletter = ic.ChannelLetter`

Put	Property is Read Only.

7.6 Base Properties

7.6.1 PropertyInt32

Integer value for property specified JSR_PropertIdEnum parameter.

Property Type	Integer
Parameters	`PropertyInt32(JSR_PropertyIDEnum)`

Get	`Dim idx as Integer`
	`idx = ic.PropertyInt32(JSR_ID_PulserDampResistorIndex)`

Put	`ic.PropertyInt32(JSR_ID_PulserDampResistorIndex) = 1`

7.6.2 PropertDouble

Double value for property specified JSR_PropertIdEnum parameter.

Property Type	Double
Parameters	`PropertDouble(JSR_PropertyIDEnum)`

Get	`Dim prf as Double`
	`prf = ic.PropertyDouble(JSR_ID_PulserPRF)`

Put	`ic.PropertyDouble(JSR_ID_PulserPRF) = 300.0`

7.6.3 PropertyBool

Bool value for property specified JSR_PropertIdEnum parameter.

Property Type	Bool
Parameters	`PropertyBool(JSR_PropertyIDEnum)`

Get	`Dim bVal as Bool`
	`bVal = ic.PropertyBool(JSR_ID_PulserIsPulsing)`

Put	`ic.PropertyBool(JSR_ID_PulserIsPulsing) = True`

7.6.4 PropertyEnum

Enumerated value for property specified JSR_PropertIdEnum parameter.

Property Type	Enumerated Value
Parameters	`PropertyEnum(JSR_PropertyIDEnum)`

Get	`Dim source as JSR_TriggerSourceEnum`
	`source = ic.PropertyEnum(JSR_TriggerSourceEnum)`

Put	`ic.PropertyEnum(JSR_TriggerSourceEnum) = JSR_TRIGGER_INTERNAL`

7.6.5 PropertyStatus

JSR_Status value for property specified JSR_PropertIdEnum parameter.

Property Type	JSR_Status
Parameters	`PropertyStatus(JSR_PropertyIDEnum)`

Get	`Dim status as JSR_Status`
	`source = ic.PropertyStatus(JSR_ID_PulserPowerLimitStatus)`

Put	`ic.PropertyStatus(JSR_ID_PulserPowerLimitStatus) = JSR_OK`

7.6.6 PropertyString

String value for property specified JSR_PropertIdEnum parameter.

Property Type	String
Parameters	`PropertyString(JSR_PropertyIDEnum)`

Get	`Dim desc as String`
	`desc = ic.PropertyString(JSR_ID_ChannelDescription)`

Put	`ic.PropertyEnum(JSR_ID_ChannelLetter) = “A”`

7.6.7 PropertyIndexValueInt32

Integer value for current index property specified JSR_PropertIdEnum parameter.

Property Type	Integer
Parameters	`PropertyIndexValueInt32(JSR_PropertyIDEnum)`

Get	`Dim val as Integer`
	`val = ic.PropertyIndexValueInt32(JSR_ID_PulserExtTriggerZIndex)`

Put	Property is Read Only.

7.6.8 PropertyIndexValueDouble

Double value for current index property specified JSR_PropertIdEnum parameter.

Property Type	Double
Parameters	`PropertyIndexValueDouble(JSR_PropertyIDEnum)`

Get	`Dim val as Double`
	`val = ic.PropertyIndexValueDouble(JSR_ID_PulserDampResistorIndex)`

Put	Property is Read Only.

7.6.9 PropertyIndexValueString

String value for current index property specified JSR_PropertIdEnum parameter.

Property Type	String
Parameters	`PropertyIndexValueString(JSR_PropertyIDEnum)`

Get	`Dim val as String`
	`val = ic.PropertyIndexValueString(JSR_ID_PulserDampResistorIndex)`

Put	Property is Read Only.

7.6.10 PropertyIndexValueByIndexInt32

Integer value for supplied index property specified JSR_PropertIdEnum parameter.

Property Type	Integer
Parameters	`PropertyIndexValueByIndexInt32(JSR_PropertyIDEnum, Integer)`

Get	`Dim val as Integer`
	`val = ic.PropertyIndexValueByIndexInt32(`
	`    JSR_ID_PulserExtTriggerZIndex,1)`

Put	Property is Read Only.

7.6.11 PropertyIndexValueByIndexDouble

Double value for supplied index property specified JSR_PropertIdEnum parameter.

Property Type	Double
Parameters	`PropertyIndexValueByIndexDouble(JSR_PropertyIDEnum, Integer)`

Get	`Dim val as Double`
	`val = ic.PropertyIndexValueByIndexDouble(`
	`    JSR_ID_PulserDampResistorIndex,1)`

Put	Property is Read Only.

7.6.12 PropertyIndexValueByIndexString

String value for supplied index property specified JSR_PropertIdEnum parameter.

Property Type	String
Parameters	`PropertyIndexValueByIndexString(JSR_PropertyIDEnum, Integer)`

Get	`Dim val as String`
	`val = ic.PropertyIndexValueByIndexString(`
	`    JSR_ID_PulserDampResistorIndex,1)`

Put	Property is Read Only.

7.6.13 PropertyMinValueInt32

Minimum Integer value for property specified JSR_PropertIdEnum parameter.

Property Type	Integer
Parameters	`PropertyMinValueInt32(JSR_PropertyIDEnum)`

Get	`Dim val as Int`
	`val = ic.PropertyMinValueInt32(JSR_ID_PulserPowerLimitVolts)`

Put	Property is Read Only.

7.6.14 PropertyMaxValueInt32

Maximum Integer value for property specified JSR_PropertIdEnum parameter.

Property Type	Integer
Parameters	`PropertyMaxValueInt32(JSR_PropertyIDEnum)`

Get	`Dim val as Int`
	`val = ic.PropertyMaxValueInt32(JSR_ID_PulserPowerLimitVolts)`

Put	Property is Read Only.

7.6.15 PropertyMinValueDouble

Minimum Double value for property specified JSR_PropertIdEnum parameter.

Property Type	Double
Parameters	`PropertyMinValueDouble(JSR_PropertyIDEnum)`

Get	`Dim val as Int`
	`val = ic.PropertyMinValueDouble(JSR_ID_PulserPRF)`

Put	Property is Read Only.

7.6.16 PropertyMaxValueDouble

Maximum Double value for property specified JSR_PropertIdEnum parameter.

Property Type	Double
Parameters	`PropertyMaxValueDouble(JSR_PropertyIDEnum)`

Get	`Dim val as Int`
	`val = ic.PropertyMaxValueDouble(JSR_ID_PulserPRF)`

Put	Property is Read Only.

7.6.17 PropertyDisplayString

Display String value for property specified JSR_PropertIdEnum parameter.

Property Type	String
Parameters	`PropertyDisplayString(JSR_PropertyIDEnum)`

Get	`Dim strprf as String`
	`strprf = ic.PropertyDipslayString(JSR_ID_PulserPRF)`

Put	Property is Read Only.

7.6.18 FormatPropertyValueInt32

Formated String given an integer value for property specified JSR_PropertIdEnum parameter.

Property Type	String
Parameters	`FormatPropertyValueInt32(JSR_PropertyIDEnum, Integer)`

Get	`Dim str as String`
	`str = ic.FormatProperyValueInt32(JSR_ID_PulserPowerLimitVolt, 100)`

Put	Property is Read Only.

7.6.19 FormatPropertyValueDouble

Formated String given a Double value for property specified JSR_PropertIdEnum parameter.

Property Type	String
Parameters	`FormatPropertyValueDouble(JSR_PropertyIDEnum, Double)`

Get	`Dim str as String`
	`str = ic.FormatProperyValueDouble(JSR_ID_PulserPRF, 1233.5)`

Put	Property is Read Only.

7.6.20 PropertyInfoName

Name for property specified JSR_PropertIdEnum parameter.

Property Type	String
Parameters	`PropertyInfoName(JSR_PropertyIDEnum)`

Get	`Dim name as String`
	`name = ic.PropertyInfoName(JSR_ID_PulserPRF)`

Put	Property is Read Only.

7.6.21 PropertyInfoDisplayName

Display name for property specified JSR_PropertIdEnum parameter.

Property Type	String
Parameters	`PropertyInfoDisplayName(JSR_PropertyIDEnum)`

Get	`Dim name as String`
	`name = ic.PropertyInfoDisplayName(JSR_ID_PulserPRF)`

Put	Property is Read Only.

7.6.22 PropertyInfoDataType

Data type for property specified JSR_PropertIdEnum parameter.

Property Type	JSR_TypeEnum
Parameters	`PropertyInfoDataType(JSR_PropertyIDEnum)`

Get	`Dim type as JSR_TypeEnum`
	`type = ic.PropertyInfoDataType(JSR_ID_PulserPRF)`

Put	Property is Read Only.

7.6.23 PropertyInfoDataTypeName

Data type display name for property specified JSR_PropertIdEnum parameter.

Property Type	String
Parameters	`PropertyInfoDataTypeName(JSR_PropertyIDEnum)`

Get	`Dim str as String`
	`str = ic.PropertyInfoDataTypeName(JSR_ID_PulserPRF)`

Put	Property is Read Only.

7.6.24 PropertyInfoAttribute

Attributes for property specified JSR_PropertIdEnum parameter.

Property Type	JSR_AttribEnum
Parameters	`PropertyInfoAttribute(JSR_PropertyIDEnum)`

Get	`Dim attr as JSR_AttribEnum`
	`attr = ic.PropertyInfoAttribute(JSR_ID_PulserPRF)`

Put	Property is Read Only.

7.7 Convenience Properties

7.7.1 JSR_ID_PulserTriggerEnable

7.7.1.1 PulserTriggerEnable

Get / Put pulser Trigger Enable for the current channel.

Property Type	`JSR_BoolEnum`

Get	`Dim State as JSR_BoolEnum`
	`State = ic.PulserTriggerEnable`

Put	`ic.PulserTriggerEnable = JSR_TRUE`

7.7.1.2 PulserTriggerEnableString

Get Pulser Trigger Enable as a string for the current channel.

Property Type	`String`

Get	`Dim ValStr as String`
	`ValStr = ic.PulserTriggerEnableString`

Put	Property is Read Only.

7.7.2 JSR_ID_PulserTriggerSource

7.7.2.1 PulserTriggerSource

Get / Put Pulser Trigger Source for the current channel.

Property Type	`JSR_TriggerSourceEnum`

Get	`Dim Source as JSR_TriggerSourceEnum`
	`Source = ic.PulserTriggerSource`

Put	Property is Read Only.

7.7.2.2 PulserTriggerSourceString

Get Pulser Trigger Source as a string for the current channel.

Property Type	`String`

Get	`Dim ValStr as String`
	`ValStr = ic.PulserTriggerSourceString`

Put	Property is Read Only.

7.7.3 JSR_ID_PulserPRF

7.7.3.1 PulserPRF

Get / Put Pulser PRF for the current channel.

Property Type	`Double`

Get	`Dim Val as Double`
	`Val = ic.PulserPRF`

Put	`ic.PulserPRF = 1234.5`

7.7.3.2 PulserPRFMin

Get minimum Pulser PRF for the current channel.

Property Type	`Double`

Get	`Dim Val as Double`
	`Val = ic.PulserPRFMin`

Put	Property is Read Only.

7.7.3.3 PulserPRFMax

Get maximum Pulser PRF for the current channel.

Property Type	`Double`

Get	`Dim Val as Double`
	`Val = ic.PulserPRFMax`

Put	Property is Read Only.

7.7.3.4 PulserPRFString

Get Pulser PRF as a string for the current channel.

Property Type	`String`

Get	`Dim ValStr as String`
	`ValStr = ic.PulserPRFString`

Put	Property is Read Only.

7.7.4 JSR_ID_PulserVolts

7.7.4.1 PulserVolts

Get / Put Pulser Volts for the current channel.

Property Type	`Double`

Get	`Dim Val as Double`
	`Val = ic.PulserVolts`

Put	`ic.PulserVolts = 100`

7.7.4.2 PulserVoltsMin

Get minimum Pulser Volts for the current channel.

Property Type	`Double`

Get	`Dim Val as Double`
	`Val = ic.PulserVoltsMin`

Put	Property is Read Only.

7.7.4.3 PulserVoltsMax

Get maximum Pulser Volts for the current channel.

Property Type	`Double`

Get	`Dim Val as Double`
	`Val = ic.PulserVoltsMax`

Put	Property is Read Only.

7.7.4.4 PulserVoltsString

Get Pulser Volts as a string for the current channel.

Property Type	`String`

Get	`Dim ValStr as String`
	`ValStr = ic.PulserVoltsString`

Put	Property is Read Only.

7.7.5 JSR_ID_PulserDampResistorIndex

7.7.5.1 PulserDampResistorIndex

Get / Put Pulser Damp Resistor Index for the current channel.

Property Type	`Integer`

Get	`Dim Idx as Integer`
	`Idx = ic.PulserDampResistorIndex`

Put	`ic.PulserVolts = 100`

7.7.5.2 PulserDampResistorIndexMin

Get minimum Pulser Damp Resistor Index for the current channel.

Property Type	`Integer`

Get	`Dim Idx as Integer`
	`Idx = ic.PulserDampResistorIndexMin`

Put	Property is Read Only.

7.7.5.3 PulserDampResistorIndexMax

Get maximum Pulser Damp Resistor Index for the current channel.

Property Type	`Integer`

Get	`Dim Idx as Integer`
	`Idx = ic.PulserDampResistorIndexMax`

Put	Property is Read Only.

7.7.5.4 PulserDampResistorOhms

Get Pulser Damp Resistor value for the current channel.

Property Type	`Double`

Get	`Dim Val as Double`
	`Val = ic.PulserDampResistorOhms`

Put	Property is Read Only.

7.7.5.5 PulserDampResistorOhmsString

Get Pulser Damp Resistor value as a string for the current channel.

Property Type	`String`

Get	`Dim ValStr as String`
	`ValStr = ic.PulserDampResistorOhmsString`

Put	Property is Read Only.

7.7.6 JSR_ID_PulserEnergyIndex

7.7.6.1 PulserEnergyIndex

Get / Put Pulser Energy Index for the current channel.

Property Type	`Integer`

Get	`Dim Idx as Integer`
	`Idx = ic.PulserEnergyIndex`

Put	`ic.PulserVolts = 100`

7.7.6.2 PulserEnergyIndexMin

Get minimum Pulser Energy Index for the current channel.

Property Type	`Integer`

Get	`Dim Idx as Integer`
	`Idx = ic.PulserEnergyIndexMin`

Put	Property is Read Only.

7.7.6.3 PulserEnergyIndexMax

Get maximum Pulser Energy Index for the current channel.

Property Type	`Integer`

Get	`Dim Idx as Integer`
	`Idx = ic.PulserEnergyIndexMax`

Put	Property is Read Only.

7.7.6.4 PulserEnergy

Get Pulser Energy Value for the current channel.

Property Type	`Double`

Get	`Dim Val as Double`
	`Val = ic.PulserEnergy`

Put	Property is Read Only.

7.7.6.5 PulserEnergyString

Get Pulser Energy Value as a string for the current channel.

Property Type	`String`

Get	`Dim ValStr as String`
	`ValStr = ic.PulserDampResistorOhmsString`

Put	Property is Read Only.

7.7.7 JSR_ID_PulserEnergyPerPulse

7.7.7.1 PulserEnergyPerPulse

Get Pulser Energy Per Pulse for the current channel.

Property Type	`Double`

Get	`Dim Val as Double`
	`Val = ic.PulserEnergyIndex`

Put	Property is Read Only.

7.7.7.2 PulserEnergyPerPulseString

Get Pulser Energy Per Pulse as a string for the current channel.

Property Type	`String`

Get	`Dim ValStr as String`
	`ValStr = ic.PulserEnergyPerPulseString`

Put	Property is Read Only.

7.7.8 JSR_ID_PulserLEDControl

7.7.8.1 PulserLEDControl

Get Pulser LED Control for the current channel.

Property Type	`JSR_PulserBlinkEnum`

Get	`Dim Val as JSR_PulserBlinkEnum`
	`Val = ic.PulserLEDControl`

Put	`ic.PulserLEDControl = JSR_LED_PULSE_ACTIVITY`

7.7.8.2 PulserLEDControlString

Get Pulser LED Control as a string for the current channel.

Property Type	`String`

Get	`Dim LedStr as String`
	`ValStr = ic.PulserLEDControlString`

Put	Property is Read Only.

7.7.9 JSR_ID_PulserIsPulsing

7.7.9.1 PulserIsPulsing

Get Pulser Is Pulsing state for the current channel.

Property Type	`JSR_BoolEnum`

Get	`Dim State as JSR_BoolEnum`
	`State = ic.PulserIsPulsing`

Put	Property is Read Only.

7.7.9.2 PulserIsPulsingString

Get Pulser Is Pulsing state as a string for the current channel.

Property Type	`String`

Get	`Dim ValStr as String`
	`ValStr = ic.PulserIsPulsingString`

Put	Property is Read Only.

7.7.10 JSR_ID_PulserPowerLimitStatus

7.7.10.1 PulserPowerLimitStatus

Get Pulser Power Limit Status for the current channel.

Property Type	`JSR_Status`

Get	`Dim Status as JSR_Status`
	`Status = ic.PulserPowerLimitStatus`

Put	Property is Read Only.

7.7.10.2 PulserPowerLimitStatusString

Get Pulser Power Limit Status as a string for the current channel.

Property Type	`String`

Get	`Dim ValStr as String`
	`ValStr = ic.PulserPowerLimitStatusString`

Put	Property is Read Only.

7.7.11 JSR_ID_PulserPowerLimitPRF

7.7.11.1 PulserPowerLimitPRF

Get Pulser Power Limit PRF for the current channel.

Property Type	`Double`

Get	`Dim Val as Double`
	`Val = ic.PulserPowerLimitPRF`

Put	Property is Read Only.

7.7.11.2 PulserPowerLimitPRFString

Get Pulser Power Limit PRF as a string for the current channel.

Property Type	`String`

Get	`Dim ValStr as String`
	`ValStr = ic.PulserPowerLimitPRFString`

Put	Property is Read Only.

7.7.12 JSR_ID_PulserPowerLimitVolts

7.7.12.1 PulserPowerLimitVolts

Get Pulser Power Limit Volts for the current channel.

Property Type	`Double`

Get	`Dim Val as Double`
	`Val = ic.PulserPowerLimitVolts`

Put	Property is Read Only.

7.7.12.2 PulserPowerLimitVoltsString

Get Pulser Power Limit Volts as a string for the current channel.

Property Type	`String`

Get	`Dim ValStr as String`
	`ValStr = ic.PulserPowerLimitVoltsString`

Put	Property is Read Only.

7.7.13 JSR_ID_PulserPowerLimitEnergyIndex

7.7.13.1 PulserPowerLimitEnergyIndex

Get / Put Pulser Power Limit Energy Index for the current channel.

Property Type	`Integer`

Get	`Dim Idx as Integer`
	`Idx = ic.PulserPowerLimitEnergyIndex`

Put	`ic.PulserPowerLimitEnergyIndex = 1`

7.7.13.2 PulserPowerLimitEnergy

Get Pulser Power Limit Energy value for the current channel.

Property Type	`Double`

Get	`Dim Val Double`
	`Val = ic.PulserPowerLimitEnergy`

Put	Property is Read Only.

7.7.13.3 PulserPowerLimitEnergyString

Get Pulser Power Limit Energy value as a string for the current channel.

Property Type	`String`

Get	`Dim ValStr as String`
	`ValStr = ic.PulserPowerLimitEnergyString`

Put	Property is Read Only.

7.7.14 JSR_ID_ReceiverBandwidth

7.7.14.1 ReceiverBandwidth

Get Receiver Bandwidth for the current channel.

Property Type	`Int32`

Get	`Dim Val as Int32`
	`Val = ic.ReceiverBandwidth`

Put	Property is Read Only.

7.7.14.2 ReceiverBandwidthString

Get Receiver Bandwidth as a string for the current channel.

Property Type	`String`

Get	`Dim BandwidthStr as String`
	`ValStr = ic.ReceiverBandwidthString`

Put	Property is Read Only.

7.7.15 JSR_ID_ReceiverSignalSelect

7.7.15.1 ReceiverSignalSelect

Get Receiver Signal Select for the current channel.

Property Type	`JSR_SignalSelectEnum`

Get	`Dim Select as JSR_SignalSelectEnum`
	`Select = ic. ReceiverSignalSelec`

Put	Property is Read Only.

7.7.15.2 ReceiverSignalSelectString

Get Receiver Signal Select as a string for the current channel.

Property Type	`String`

Get	`Dim ValStr as String`
	`ValStr = ic.ReceiverSignalSelectString`

Put	Property is Read Only.

7.7.16 JSR_ID_ReceiverGainDB

7.7.16.1 ReceiverGainDB

Get / Put Receiver Gain in dB for the current channel.

Property Type	`Double`

Get	`Dim Gain as Double`
	`Gain = ic.ReceiverGainDB`

Put	`ic.ReceiverGainDB = 11`

7.7.16.2 ReceiverGainDBMin

Get minimum Receiver Gain in dB for the current channel.

Property Type	`Double`

Get	`Dim Gain as Double`
	`Gain = ic.ReceiverGainDBMin`

Put	Property is Read Only.

7.7.16.3 ReceiverGainDBMax

Get maximum Receiver Gain in dB for the current channel.

Property Type	`Double`

Get	`Dim Gain as Double`
	`Gain = ic.ReceiverGainDBMax`

Put	Property is Read Only.

7.7.16.4 ReceiverGainDBString

Get Receiver Gain as a string for the current channel.

Property Type	`String`

Get	`Dim ValStr as String`
	`ValStr = ic.ReceiverGainDBString`

Put	Property is Read Only.

7.7.17 JSR_ID_ReceiverThroughGainDB

7.7.17.1 ReceiverThroughGainDB

Get / Put Receiver Through Gain in dB for the current channel.

Only the PRC50 supports this property.

Property Type	`Double`

Get	`Dim Gain as Double`
	`Gain = ic.ReceiverThroughGainDB`

Put	`ic.ReceiverThroughGainDB = 11`

7.7.17.2 ReceiverThroughGainDBMin

Get minimum Receiver Through Gain in dB for the current channel.

Only the PRC50 supports this property.

Property Type	`Double`

Get	`Dim Gain as Double`
	`Gain = ic.ReceiverThroughGainDBMin`

Put	Property is Read Only.

7.7.17.3 ReceiverThroughGainDBMax

Get maximum Receiver Through Gain in dB for the current channel.

Only the PRC50 supports this property.

Property Type	`Double`

Get	`Dim Gain as Double`
	`Gain = ic.ReceiverThroughGainDBMax`

Put	Property is Read Only.

7.7.17.4 ReceiverThroughGainDBString

Get Receiver Through Gain as a string for the current channel.

Only the PRC50 supports this property.

Property Type	`String`

Get	`Dim ValStr as String`
	`ValStr = ic.ReceiverThroughGainDBString`

Put	Property is Read Only.

7.7.18 JSR_ID_ReceiverTREchoGainDB

7.7.18.1 ReceiverTREchoGainDB

Get / Put Receiver TR Echo Gain in dB for the current channel.

Only the PRC50 supports this property.

Property Type	`Double`

Get	`Dim Gain as Double`
	`Gain = ic.ReceiverTREchoGainDB`

Put	`ic.ReceiverTREchoGainDB = 11`

7.7.18.2 ReceiverTREchoGainDBMin

Get minimum Receiver TR Echo Gain in dB for the current channel.

Only the PRC50 supports this property.

Property Type	`Double`

Get	`Dim Gain as Double`
	`Gain = ic.ReceiverTREchoGainDBMin`

Put	Property is Read Only.

7.7.18.3 ReceiverTREchoGainDBMax

Get maximum Receiver TR Echo Gain in dB for the current channel.

Only the PRC50 supports this property.

Property Type	`Double`

Get	`Dim Gain as Double`
	`Gain = ic.ReceiverTREchoGainDBMax`

Put	Property is Read Only.

7.7.18.4 ReceiverTREchoGainDBString

Get Receiver TR Echo Gain in dB as a string for the current channel.

Only the PRC50 supports this property.

Property Type	`String`

Get	`Dim ValStr as String`
	`ValStr = ic.ReceiverTREchoGainDBString`

Put	Property is Read Only.

7.7.19 JSR_ID_ReceiverLPFilterIndex

7.7.19.1 ReceiverLPFilterIndex

Get / Put Receiver Low Pass Filter Index for the current channel.

Property Type	`Integer`

Get	`Dim Idx as Integer`
	`Idx = ic.ReceiverLPFilterIndex`

Put	`ic.ReceiverLPFilterIndex = 1`

7.7.19.2 ReceiverLPFilterIndexMin

Get minimum Receiver Low Pass Filter Index for the current channel.

Property Type	`Integer`

Get	`Dim Idx as Integer`
	`Idx = ic.ReceiverLPFilterIndexMin`

Put	Property is Read Only.

7.7.19.3 ReceiverLPFilterIndexMax

Get maximum Receiver Low Pass Filter Index for the current channel.

Property Type	`Integer`

Get	`Dim Idx as Integer`
	`Idx = ic.ReceiverLPFilterIndexMax`

Put	Property is Read Only.

7.7.19.4 ReceiverLPFilterMHz

Get Receiver Low Pass Filter in MHz for the current channel.

Property Type	`Double`

Get	`Dim Val as Double`
	`Val = ic.ReceiverLPFilterMHz`

Put	Property is Read Only.

7.7.19.5 ReceiverLPFilterMHzString

Get Receiver Low Pass Filter in MHz as a string for the current channel.

Property Type	`String`

Get	`Dim ValStr as String`
	`ValStr = ic.ReceiverLPFilterMHzString`

Put	Property is Read Only.

7.7.20 JSR_ID_GetReceiverHPFilterIndex

7.7.20.1 ReceiverHPFilterIndex

Get / Put Receiver High Pass Filter Index for the current channel.

Property Type	`Integer`

Get	`Dim Idx as Integer`
	`Idx = ic.ReceiverHPFilterIndex`

Put	`ic.ReceiverHPFilterIndex = 1`

7.7.20.2 ReceiverHPFilterIndexMin

Get minimum Receiver High Pass Filter Index for the current channel.

Property Type	`Integer`

Get	`Dim Idx as Integer`
	`Idx = ic.ReceiverHPFilterIndexMin`

Put	Property is Read Only.

7.7.20.3 ReceiverHPFilterIndexMax

Get maximum Receiver High Pass Filter Index for the current channel.

Property Type	`Integer`

Get	`Dim Idx as Integer`
	`Idx = ic.ReceiverHPFilterIndexMax`

Put	Property is Read Only.

7.7.20.4 ReceiverHPFilterMHz

Get Receiver High Pass Filter in MHz for the current channel.

Property Type	`Double`

Get	`Dim Val as Double`
	`Val = ic.ReceiverHPFilterMHz`

Put	Property is Read Only.

7.7.20.5 ReceiverHPFilterMHzString

Get Receiver High Pass Filter in MHz as a string for the current channel.

Property Type	`String`

Get	`Dim ValStr as String`
	`ValStr = ic.ReceiverHPFilterMHzString`

Put	Property is Read Only.

7.8 JSR_Simple_ActiveX.DLL

7.8.1 Registering DLL

The JSR_Simple_AxtiveX.dll must be registered with “Windows” OS. The installation process of the SDK should perform this registation.

Note: End user application installations require this registration process, so the installer you distribute must perform the registration.

Manual method of registration:

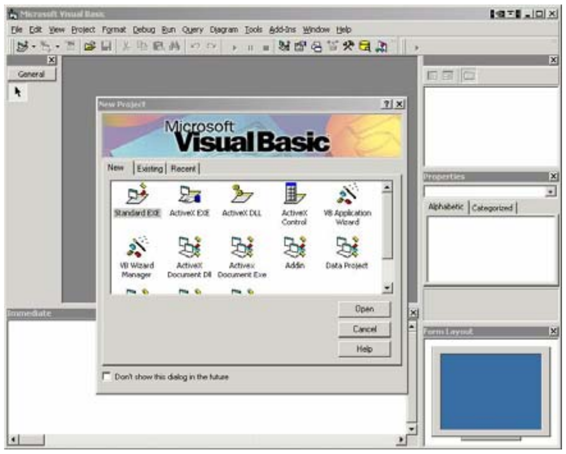
```
C:> regsvr32 %dir%/JSR_Simple_ActiveX.dll
```

7.8.2 Visual Basic Reference of JSR_Simple_ActiveX.dll

Create a new Visual Basic Project and Reference the JSR_Simple_ActiveX.dll.

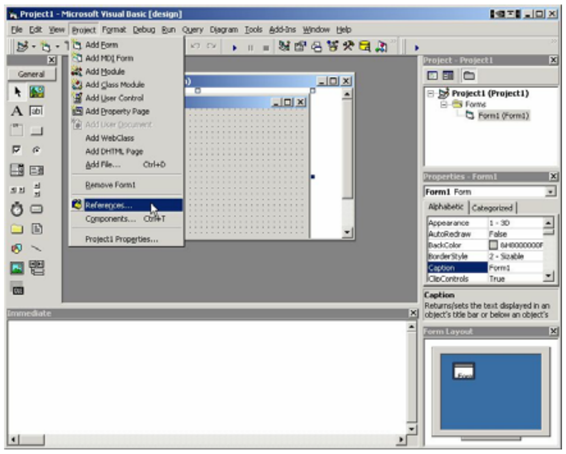
7.8.2.1 New Visual Basic Project

Create a new Visual Basic project.



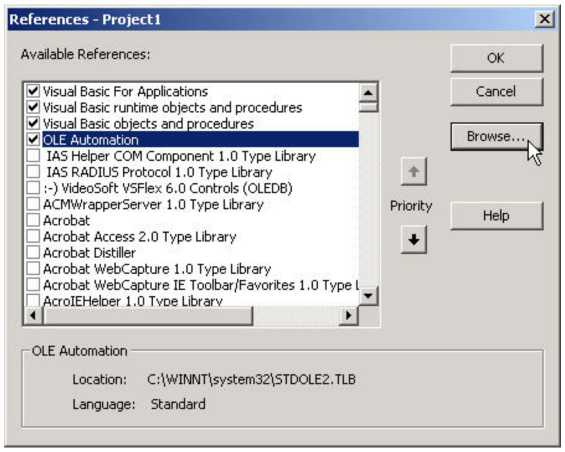
7.8.2.2 Open Menu / Project / References… Dialog

Using the Menu, select “Project” and then “References…” menu item to open the References dialog.



7.8.2.3 References Dialog Browse

Press “Browse…” button to open the “Add Reference” dialog



Available References:

- [✓] Visual Basic For Applications
- [✓] Visual Basic runtime objects and procedures
- [✓] Visual Basic objects and procedures
- [✓] OLE Automation
- [ ] IAS Helper COM Component 1.0 Type Library
- [ ] IAS RADIUS Protocol 1.0 Type Library
- [ ] :-) VideoSoft VSFlex 6.0 Controls (OLEDB)
- [ ] ACMWrapperServer 1.0 Type Library
- [ ] Acrobat
- [ ] Acrobat Access 2.0 Type Library
- [ ] Acrobat Distiller
- [ ] Acrobat WebCapture 1.0 Type Library
- [ ] Acrobat WebCapture IE Toolbar/Favorites 1.0 Type L
- [ ] AcroIEHelper 1.0 Type Library

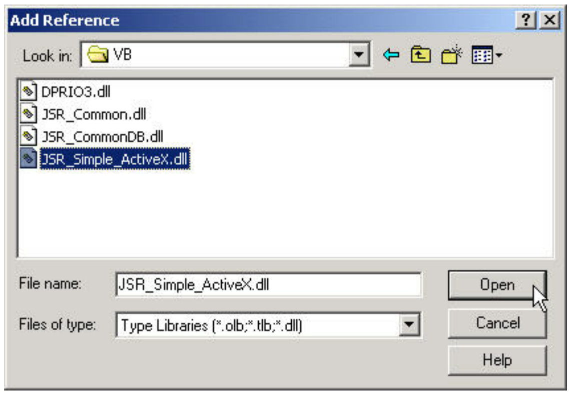
OLE Automation

Location: C:\WINNT\system32\STDOLE2.TLB

Language: Standard

7.8.2.4 Add Reference

Select the JSR_Simple_ActiveX.dll file and press the “Open” button.



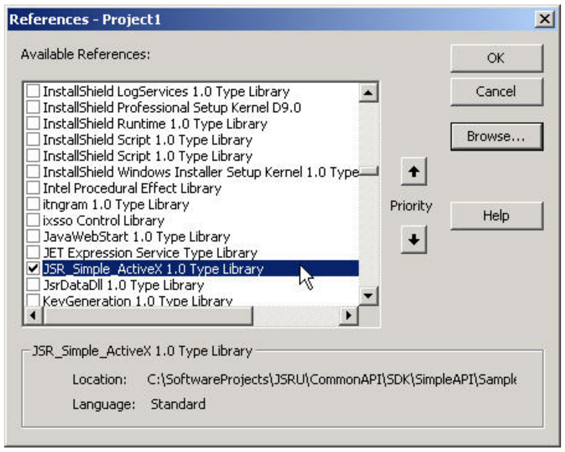
DPRIO3.dll
JSR_Common.dll
JSR_CommonDB.dll
JSR_Simple_ActiveX.dll

File name: JSR_Simple_ActiveX.dll

Files of type: Type Libraries (*.olb;*.tlb;*.dll)

7.8.2.5 Reference Added

The Visual Basic Project now references the JSR_Simple_ActiveX library.



Available References:

- [ ] InstallShield LogServices 1.0 Type Library
- [ ] InstallShield Professional Setup Kernel D9.0
- [ ] InstallShield Runtime 1.0 Type Library
- [ ] InstallShield Script 1.0 Type Library
- [ ] InstallShield Script 1.0 Type Library
- [ ] InstallShield Windows Installer Setup Kernel 1.0 Type
- [ ] Intel Procedural Effect Library
- [ ] itngram 1.0 Type Library
- [ ] ixsso Control Library
- [ ] JavaWebStart 1.0 Type Library
- [ ] JET Expression Service Type Library
- [✓] JSR_Simple_ActiveX 1.0 Type Library
- [ ] JsrDataDll 1.0 Type Library
- [ ] KeyGeneration 1.0 Type Library

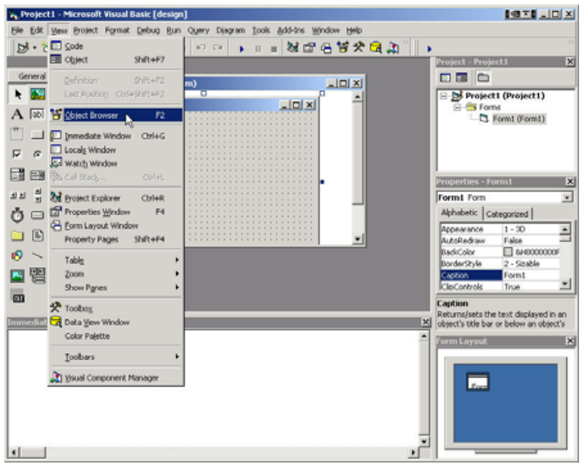
JSR_Simple_ActiveX 1.0 Type Library

Location: C:\SoftwareProjects\JSRU\CommonAPI\SDK\SimpleAPI\Sample

Language: Standard

7.8.2.6 Menu View / Object Browser…

Verify that the JSR_Simple_ActiveX objects are available by opening the “Object Browser” window.

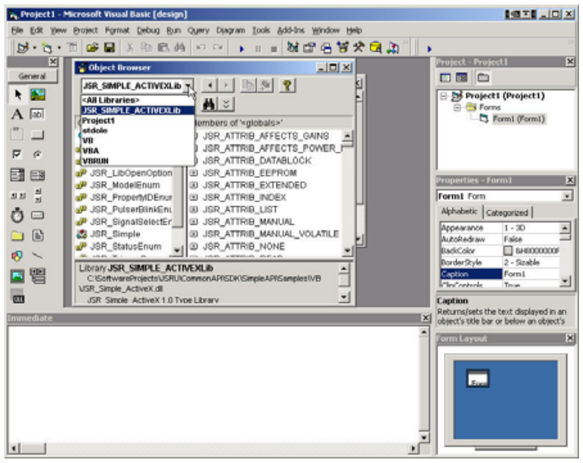


Appearance	1 - 3D
AutoRedraw	False
BackColor	&H8000000F
BorderStyle	2 - Sizable
Caption	Form1
ClipControls	True

Caption
Returns/sets the text displayed in an object's title bar or below an object's

7.8.2.7 Object Browser

Select JSR_SIMPLE_ACTIVEXLib in the “Object Browser” window to view members and functions available.



JSR_LibOpenOption
JSR_ModelEnum
JSR_PropertyIDEnur
JSR_PulserBlinkEnu
JSR_SignalSelectEr
JSR_Simple
JSR_StatusEnum

Members of '<globals>'
JSR_ATTRIB_AFFECTS_GAINS
JSR_ATTRIB_AFFECTS_POWER_
JSR_ATTRIB_DATABLOCK
JSR_ATTRIB_EEPROM
JSR_ATTRIB_EXTENDED
JSR_ATTRIB_INDEX
JSR_ATTRIB_LIST
JSR_ATTRIB_MANUAL
JSR_ATTRIB_MANUAL_VOLATILE
JSR_ATTRIB_NONE

Library **JSR_SIMPLE_ACTIVEXLib**
C:\SoftwareProjects\JSRU\CommonAPISDK\SimpleAPISamples\VB\JSR_Simple_ActiveX.dll
JSR_Simple_ActiveX 1.0 Type Library

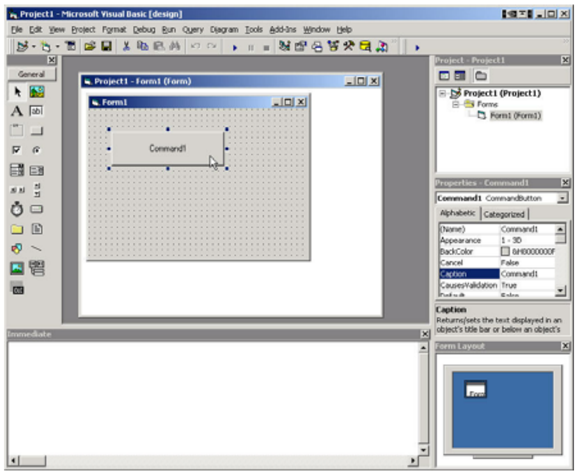
Appearance	1 - 3D
AutoRedraw	False
BackColor	&H8000000F
BorderStyle	2 - Sizable
Caption	Form1
ClipControls	True

Caption
Returns/sets the text displayed in an object's title bar or below an object's

7.8.3 Sample Visual Basic Project

7.8.3.1 Create a “Command” button

Create a command button by clicking the “Button” from the object palette and then creating a button on the “Form”.

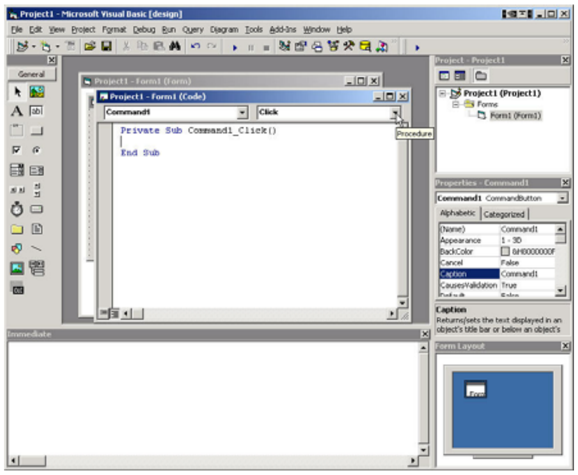


(Name)	Command1
Appearance	1 - 3D
BackColor	&H8000000F
Cancel	False
Caption	Command1
CausesValidation	True

Caption
Returns/sets the text displayed in an object's title bar or below an object's

7.8.3.2 Create a “Click” procedure

Double click the “Command” button to create a “Click” procedure.



```
Private Sub Command1_Click()

End Sub
```

7.8.3.3 Sample “Click” procedure

Sample code for the “Click” procedure of the “Command” button:

```
Private Sub Command1_Click()
Dim ic As New JSR_Simple

'Open library in simulation mode
Call ic.Open(JSR_LIB_OPTION_SIMULATE, JSR_MODEL_ANY)

'Channel Count
Debug.Print "ic.ChannelCount = ", ic.ChannelCount

'Channel has not been set yet, will produce -1
Debug.Print "ic.Channel = ", ic.Channel

'Setting the channel to 0
ic.Channel = 0
Debug.Print "ic.Channel = ", ic.Channel

' Channel Description
Debug.Print "ic.ChannelDecription = ", ic.ChannelDescription

'Property Information for JSR_ID_PulserPRF
Debug.Print "ic.PropertyInfoName(JSR_ID_PulserPRF) = ",
ic.PropertyInfoName(JSR_ID_PulserPRF)
Debug.Print "ic.PropertyInfoDisplayName(JSR_ID_PulserPRF) = ",
ic.PropertyInfoDisplayName(JSR_ID_PulserPRF)
Debug.Print "ic.PropertyInfoDataType(JSR_ID_PulserPRF) = ",
ic.PropertyInfoDataType(JSR_ID_PulserPRF)
Debug.Print "ic.PropertyInfoDataTypeName(JSR_ID_PulserPRF) = ",
ic.PropertyInfoDataTypeName(JSR_ID_PulserPRF)

'Using Base Properties
Debug.Print "ic.PropertyDouble(JSR_ID_PulserPRF) = ", ic.PropertyDouble(JSR_ID_PulserPRF)
ic.PropertyDouble(JSR_ID_PulserPRF) = 300
Debug.Print "ic.PropertyDouble(JSR_ID_PulserPRF) = ", ic.PropertyDouble(JSR_ID_PulserPRF)
Debug.Print "ic.PropertyDisplayString(JSR_ID_PulserPRF) = ",
ic.PropertyDisplayString(JSR_ID_PulserPRF)

'Using Convenience Properties
Debug.Print "ic.PulserPRF = ", ic.PulserPRF
ic.PulserPRF = 500
Debug.Print "ic.PulserPRF = ", ic.PulserPRF
Debug.Print "ic.PulserPRFString = ", ic.PulserPRFString

Call ic.Close
End Sub
```

7.8.3.4 Sample Output

Sample output from running the project and pressing the “Command” Button:

```
ic.ChannelCount =            3
ic.Channel =  -1
ic.Channel =   0
ic.ChannelDecription =      Sim PRC50, PCI Slot 0, 50 MHz
ic.PropertyInfoName(JSR_ID_PulserPRF) =   JSR_ID_PulserPRF
ic.PropertyInfoDisplayName(JSR_ID_PulserPRF) =          PRF
ic.PropertyInfoDataType(JSR_ID_PulserPRF) =              1
ic.PropertyInfoDataTypeName(JSR_ID_PulserPRF) =         Double
ic.PropertyDouble(JSR_ID_PulserPRF) =      100
ic.PropertyDouble(JSR_ID_PulserPRF) =      300
ic.PropertyDisplayString(JSR_ID_PulserPRF) =            300 Hz
ic.PulserPRF =               300
ic.PulserPRF =               500
ic.PulserPRFString =        500 Hz
```