JSR Common API Library
Software Development Kit

Properties Reference

SDK Version 1.3.0.0
Updated March, 2011

Imaginant Inc.
www.JSRUltrasonics.com

TechSupport@JSRUltrasonics.com

Table of Contents

1 Scope

This document is a list of Property identification enumerations, “Property ID’s” in the JSR Common API Library (JSR Library herein). Property values, settings, data types, and other relevant information, as well as notes on how to use them. This document is designed to be a reference to the Properties only, it is not a complete overview of the JSR Library.

The JSR Common API Library is based on an interface model of Objects with Properties, which is explained in detail in JSR_API_ProgrammersReferenceManual.pdf. This document should be used with the following documents:

JSR_API_ProgrammersReferenceManual.pdf
- Documentation on the JSR Library as a whole

JSR_Common.h
- Header file declaring the C interface to the JSR Common Library

JSR_Types.h
- Header file of types and structs used in JSR Common Library

JSR_Status.h
- Header containing status codes, and status tools

JSR_PropertyID.h
- Header containing the complete Property list

JSR_Loader.h
- Header for performing run-time loading of the DLL

JSR_Loader.c
- Source code for performing run-time loading of the DLL

2 Property Overview

The JSR Library is a collection of Objects, with each Object having a set of Properties. Properties are the only way to get or set data to an Object. Each property has a 32bit integer that is used as an ID, and these ID's are assigned by ranges, based on the Property's type and use.

The ranges are broken down as follows:

Property Type	**Property Sub-Type**	**Property ID Sub-Type Range**	**Property ID Range**
Standard Properties		**1 - 499**	1 - 499
Reference Properties		500 - 999	500 - 999
Library Object Properties	Baseline Library Properties	1000 - 1499	1000 - 1999
	Extended Library Properties (none)	1500 - 1999	
Instrument Object Properties	Baseline Instrument Properties	2000 - 2499	2000 – 2999
	Extended Instrument Properties	2500 - 2999	
Channel Object Properties	Baseline Channel Properties	3000 - 3499	3000 - 3999
	Extended Channel Properties	3500 - 3999	
Pulser Object Properties	Baseline Pulser Properties	4000 - 4499	4000 - 4999
	Extended Pulser Properties	4500 - 4999	
Receiver Object Properties	Baseline Receiver Properties	5000 - 5499	5000 – 5999
	Extended Receiver Properties	5500 - 5999	

3 Property Attributes

The JSR_InfoStruct and JSR_AsciiInfoStruct for every property has a field named “attribs”. The attribs field is a bit mask that specifies how that property should be used:

JSR_ATTRIB_NONE	This is zero, and should never occur. If your application finds this as the attrib value, that indicates the call to JSR_GetInfo() or JSR_GetAsciiInfo() failed. The failure should have been detected by your code as the JSR_Status value returned by the call to JSR_GetInfo() or JSR_GetAsciiInfo(). If the attribs field contains this value, all other fields of the infoStruct are invalid.
JSR_ATTRIB_READ	The property can be read with a “Get” function call. If no other attribute bits are set, then this is a read-only property; attempting a “Set” to this property will cause an error return.
JSR_ATTRIB_WRITE	The property can be written to with a “Set” function call and is intended to be used by the application to control behavior of the instrument. Properties that can be “Set” will also have the JSR_ATTRIB_READ bit set.
JSR_ATTRIB_VOLATILE	The value of this property may change based on either the setting of some other property or some external event. The “Power Limit” properties are an example of properties that can change based on settings to other properties. This attribute bit is also set if the connected DPR300 instrument has manual front-panel controls which would allow the operator to change the property settings without the host software.
JSR_ATTRIB_LIST	This attribute is a “list” or an array intended to be used with an index value. The base data type of this list can be any of the JSR data types. Your application should not assume the data type of a list, but should examine the elementType field of the JSR_InfoStruct or JSR_AsciiInfoStruct returned by a call to JSR_GetInfo() or JSR_GetAsciiInfo().
JSR_ATTRIB_INDEX	This attribute is an “index” into either an array of numbers or an enumeration. The list associated with the value is contained in a separate property. That separate property is identified in the “listID” field of the JSR_InfoStruct and JSR_AsciiInfoStruct for this property.
JSR_ATTRIB_MANUAL	This attribute bit is set if the connected DPR300 instrument has manual front-panel controls which would allow the operator to change this property setting without the host software.
JSR_ATTRIB_EXTENDED	This attribute bit is set if the property is an “extended” property and therefore may only be present for one particular instrument model or some subset of all instrument models. If you want your host application to be able to operate all models of JSR instruments, including future products, your application should gracefully handle the case of this property not being present. If the selected instrument does not contain this property, any of the function calls to access this property will return an error status.
JSR_ATTRIB_DATABLOCK	This is for internal JSR use only. Host applications should ignore this attribute bit.

JSR_ATTRIB_REFERENCE	This bit indicates the property is a “Reference” property which has special behavior: • All Reference Properties are available from all valid object handles. • The Reference properties for all objects are identical. • The Reference properties are always read-only.
JSR_ATTRIB_AFFECTS_POWER_LIMIT	This bit indicates the setting of this property will control the values read from the JSR Common Library for any of the “Power Limit” attributes. See JSR_ID_PulserPowerLimitStatus. Power Limit is a non-linear function of two or three interacting property settings. Different instruments and different remote pulsers may affect the Power Limit differently, so application behavior should be based on run-time sampling the power limit properties after the application has changed any property that has this attribute. If a property has this bit set, your host application should test the JSR_ID_PulserPowerLimitStatus after setting a new value to this property in order to properly report the updated status to the user.
JSR_ATTRIB_AFFECTS_GAINS	This attribute bit will be set for properties that affect the receiver gain if the selected instrument has more than one control property that can affect the receiver gain.
JSR_ATTRIB_SUPERSEDED	This attribute bit will be set for a baseline property that has a similar extended property or properties that allow more specific and/or more complex instrument control. This can be used to ‘deselect’ baseline properties for display if more in-depth extended properties exist. The baseline property can still be used for control, despite the ‘better’ extended properties. Use of both the more in-depth extended properties and the superseded control in the same program is discouraged, but possible. For example, on PRC50 instruments this attribute is set for baseline property JSR_ID_ReceiverGainDB, because the PRC50 also has the extended properties JSR_ID_ReceiverTREchoGainDB and JSR_ID_ReceiverThroughGainDB which allow separate gai control of the two amplifier paths.
JSR_ATTRIB_EEPROM	This attribute bit indicates properties that are stored the EEPROM in the instrument. Application code may ignore this attribute.
Other attributes in JSR_AttribEnum	The other enumerations listed in JSR_Types.h JSR_AttribEnum are common combinations of the attributes defined above, and exist for easier application writing and debugging. Some uncommon combinations do not have a specific named enumeration, but application code can either test the one-bit attributes separately or logically combine attributes for comparison.

4 Standard Properties

Standard Properties are those Properties that are present in every Object in the JSR Library.

Standard Properties have ID’s in the range 1 to 499.

Each and every JSR Library object contains every Standard Property, as well as any baseline and extended properties that are unique to the specific object.

4.1 JSR_ID_AvailablePropertyIDs

Data Type	Count	Attributes	Object
JSR_PropID	Variable	Read	ALL

This property contains an array of JSR_PropID values. This array is a list of each and every property that is valid for the specified object. These properties may come from three different sets of properties (Standard, the Object's Baseline properties, and the Object's Extended properties). Therefore, there will gaps between some of the ID values in the array. The list of available property ID's is determined at run time before an object is opened. Once an object is opened, its list of available ID's will remain constant.

The list of available ID's will be different from object to object, and may be different from instrument to instrument. Individual instruments may have slightly different lists of available properties based on the configuration of the instrument. For example, two DPR500 objects may have different properties based on the type of Remote Pulsers installed.

NOTE: Reference properties are available via ANY opened object, but are only included in this list of JSR_ID_AvailablePropertyIDs for the Library Object. This is because requests for Reference ID's are all 'caught' and handled by the Library on behalf of all objects. This also prevents redundant loading of reference properties by 3rd party applications.

Newer releases of the JSR Common API Library may have additional properties not available in earlier releases. This is important to keep in mind when designing a system to be forward compatible for new versions of the JSR Common API library. Properly using JSR_ID_AvailablePropertyIDs for generating property lists will allow your application to be upgraded by a simple JSRCommon.DLL upgrade, without recompiling the entire program.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_PropertyIDEnum nPropIdArray[n];
```

Prototypes

```
JSR_Status JSR_GetInt32(nHandle, JSR_ID_AvailablePropertyIDs, n, nPropIdArray);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_PropertyIDEnum nPropIdArray[n];
```

Prototypes

```
JSR_Status JSRLibFuncs.GetInt32(nHandle, JSR_ID_AvailablePropertyIDs, n, nPropIdArray);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_PropertyIDEnum nPropIdArray[n];
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetArrayInt32(JSR_ID_AvailablePropertyIDs,(JSR_Int32*) nPropIdArray);
```

JSR_Simple_ActiveX.dll

<Not Implemented>

4.2 JSR_ID_FirstIDNumber

Data Type	Count	Attributes	Object
JSR_PropID	1	Read	ALL

JSR Ultrasonics recommends using the property JSR_ID_AvailablePropertyIDs rather than this one.

Returns the JSR_PropID of the first Property available in an Object specific to that object.

Note that this does **not** include Standard properties or Reference properties. Standard and Reference properties are excluded, since they are constant across all Objects.

This value can be used with JSR_ID_LastIDNumber to loop through all Object-specific properties for testing, or automatic interface/data investigation. See the section 'Auto-Population of Object data via Properties' for more information on how to do this quickly and efficiently.

JSR_Common.lib

Declarations

```
JSR_PropertyIDEnum nPropId;
```

Prototype

```
JSR_Status JSR_GetInt32(nHandle, JSR_ID_FirstIdNumber, 1, (JSR_Int32*)&nPropId);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_PropertyIDEnum nPropId;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetInt32(nHandle, JSR_ID_FirstIdNumber, 1, (JSR_Int32*)&nPropId);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_PropertyIDEnum nPropId;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetInt32(JSR_ID_FirstIdNumber, (JSR_Int32*)&nPropId);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM nPropId as JSR_PropertyIDEnum
```

Specific Prototypes

<Not Implemented>

Genric Prototypes

```
nPropId = ic.PropertyInt32(JSR_ID_FirstIdNumber)
```

4.3 JSR_ID_LastIDNumber

Data Type	Count	Attributes	Object
JSR_PropID	1	Read	ALL

JSR Ultrasonics recommends using the property JSR_ID_AvailablePropertyIDs rather than this one.

Returns the JSR_PropID of the last Property available in an Object specific to that object..

This value can be used with JSR_ID_FirstIDNumber to loop through all Object-specific properties for testing, or automatic interface/data investigation. See the section 'Auto-Population of Object data via Properties' for more information on how to do this quickly and efficiently.

JSR_Common.lib

Declarations

```
JSR_PropID nPropId;
```

Prototype

```
JSR_Status JSR_GetInt32(nHandle, JSR_ID_LastIdNumber, 1, (JSR_Int32*)&nPropId);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_PropID nPropId;
```

Prototype

```
JSR_Status JSRLibFuncs.GetInt32(nHandle, JSR_ID_LastIdNumber, 1, (JSR_Int32*)&nPropId);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_PropID nPropId;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetInt32(JSR_ID_LastIdNumber, (JSR_Int32*)&nPropId);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM nPropId as JSR_PropertyIDEnum
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
nPropId=ic.PropertyInt32(JSR_ID_LastIdNumber)
```

4.4 JSR_ID_ParentHandle

Data Type	Count	Attributes	Object
JSR_Handle	1	Read	ALL

Returns the handle of the Object that is the 'Parent' of the Object. The parent of an object is another Object, which is the object that 'contains' it. Parenthood is a well defined relationship. The following list shows objects and their parent type.

Object type	Parent Object type
Library	Library (the library itself, by definition)
Instrument	Library
Channel	Instrument
Pulser	Channel
Receiver	Channel

JSR_Common.lib

Declarations

```
JSR_Handle nParentHandle, nHandle;
```

Prototype

```
JSR_Status JSR_GetInt32(nHandle, JSR_ID_ParentHandle, 1, (JSR_Int32*)&nParentHandle);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nParentHandle, nHandle;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetInt32(nHandle, JSR_ID_ParentHandle, 1, (JSR_Int32*)&nParentHandle);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Handle nParentHandle, nHandle;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetInt32(nHandle, JSR_ID_ParentHandle, (JSR_Int32*)&nParentHandle);
```

JSR_Simple_ActiveX.dll

<Not Implemented>

4.5 JSR_ID_ObjectType

Data Type	Count	Attributes	Object
JSR_Enum	1	Read	ALL

Returns the enumerated value that identifies which type of object this object is. The possible values for this ID can be found in JSR_Types.h, in the Enum JSR_ObjTypeEnum. This ID can be used with the property id JSR_ID_JSR_ID_ReferenceObjectTypeNames to get the plain text name of an object for display. See the section 'Using Enum's and name ID's'

See JSR_ID_ReferenceObjectTypeNames

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_ObjTypeEnum nObjType;
```

Prototypes

```
JSR_Status JSR_GetInt32(nHandle, JSR_ID_ObjectType, 1, (JSR_Int32*)&nObjType);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_ObjTypeEnum nObjType;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetInt32(nHandle, JSR_ID_ObjectType, 1, (JSR_Int32*)&nObjType);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Handle nHandle;
JSR_ObjTypeEnum nObjType;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetInt32(nHandle, JSR_ID_ObjectType, (JSR_Int32*)& nObjType);
```

JSR_Simple_ActiveX.dll

<Not Implemented>

5 Reference Properties accessible from all objects

Reference properties are always read-only and can be accessed using any valid object handle once the object is open. Reference properties are completely identical and always have the same data no matter which object handle is used to access them.

You may choose to have your application load all the reference properties into your own library object or load only those needed by each lower object into those lower objects as needed by each object. Your application can determine which reference properties are needed by each object by examining the “listID” field of each property contained by the specific object.

The concept of baseline properties vs. extended properties does not apply to reference properties.

5.1 JSR_ID_ReferenceModelNames

Data Type	Count	Attributes	Object
JSR_String or JSR_Ascii	Variable	Read	All

This property contains a list of names of instrument models that correspond to the values in JSR_ModelEnum.

Enum	String	Index
JSR_MODEL_UNKNOWN	“Unknown”	0
JSR_MODEL_ANY	“Any”	1
JSR_MODEL_PRC50	“PRC50”	2
JSR_MODEL_DPR500	“DPR500”	3
JSR_MODEL_DPR300	“DPR300”	4

When a new version of the JSR Common API Library is released that supports new instrument models, the JSR_ModelEnum list will contain new values enumeration, and this property will return additional names.

This property reports the names of all the models supported by the JSR Common DLL. Every model is always listed, even if the underlying drivers are not installed, and even if that model had not been requested by the application’s call to JSR_OpenLibrary().

This property is the listID property for properties JSR_ID_LibraryDriversStatus, JSR_ID_LibrarySupportedModels, and JSR_ID_InstrumentModelEnum.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_String sStringArray[n];
JSR_Ascii sAsciiArray[n];
```

Prototypes

```
JSR_Status JSR_GetString(nHandle, JSR_ID_ReferenceModelNames, n, sStringArray);
JSR_Status JSR_GetAscii(nHandle, JSR_ID_ReferenceModelNames, n, sAsciiArray);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_String sStringArray[n];
JSR_Ascii sAsciiArray[n];
```

Prototypes

```
JSR_Status JSRLibFuncs.GetString(nHandle, JSR_ID_ReferenceModelNames, n, sStringArray);
JSR_Status JSRLibFuncs.GetAscii(nHandle, JSR_ID_ReferenceModelNames, n, sAsciiArray);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_String sStringArray[n], *pStringArray;
JSR_Ascii sAsciiArray[n], *pAsciiArray;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
// You must allocate buffers before calling
JSR_Status ic.GetArrayString(JSR_ID_ReferenceModelNames, n, (JSR_String*)sStringArray);
JSR_Status ic.GetStringArray(JSR_ID_ReferenceModelNames, n, (JSR_Ascii*)sAsciiArray);

// Returns allocated strings that must eventually be freed
JSR_Status ic.GetArrayString(JSR_ID_ReferenceModelNames, (JSR_String**)&pStringArray);
JSR_Status ic.GetArrayString(JSR_ID_ReferenceModelNames, (JSR_Ascii**)&pAsciiArray);
```

JSR_Simple_ActiveX.dll

<Not Implemented>

5.2 JSR_ID_ReferenceDataTypeNames

Data Type	Count	Attributes	Object
JSR_String or JSR_Ascii	Variable	Read	All

This property contains a set of strings containing Data Type names that correspond to the values in JSR_TypeEnum. Although we have tried to think of all possible data types for future versions of the JSR Common SDK, it is possible that some future version may have more enumerations in JSR_TypeEnum.

The strings for the JSR_TypeEnum's may be useful to an application developer while creating and debugging application code but are probably not at all relevant to the end user of your application or system. There is usually no need to present the names of these enum's in a GUI.

We highly recommend that any place your application code has a switch statement based on a variable of type JSR_TypeEnum, that the code have a "default:" clause to gracefully handle future situations where the enumeration list is larger than you expected. See the section of this document 'Using Enum's and Name Property ID's' for how to use this enum for display purposes.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_String sStringArray[n];
JSR_Ascii sAsciiArray[n];
```

Prototypes

```
JSR_Status JSR_GetString(nHandle, JSR_ID_ReferenceDataTypeNames, n, sStringArray);
JSR_Status JSR_GetAscii(nHandle, JSR_ID_ReferenceDataTypeNames, n, sAsciiArray);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_String sStringArray[n];
JSR_Ascii sAsciiArray[n];
```

Prototypes

```
JSR_Status JSRLibFuncs.GetString(nHandle, JSR_ID_ReferenceDataTypeNames, n, sStringArray);
JSR_Status JSRLibFuncs.GetAscii(nHandle, JSR_ID_ReferenceDataTypeNames, n, sAsciiArray);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_String sStringArray[n], *pStringArray;
JSR_Ascii sAsciiArray[n], *pAsciiArray;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
// You must allocate buffers before calling
JSR_Status ic.GetArrayString(JSR_ID_ReferenceDataTypeNames, n, (JSR_String*)sStringArray);
JSR_Status ic.GetArrayString(JSR_ID_ReferenceDataTypeNames, n, (JSR_Ascii*)sAsciiArray);

// Returns allocated strings that must eventually be freed
JSR_Status ic.GetArrayString(JSR_ID_ReferenceDataTypeNames, (JSR_String**)&pStringArray);
JSR_Status ic.GetArrayString(JSR_ID_ReferenceDataTypeNames, (JSR_Ascii**)&pAsciiArray);
```

JSR_Simple_ActiveX.dll

<Not Implemented>

5.3 JSR_ID_ReferenceUnitNames

Data Type	Count	Attributes	Object
JSR_String or JSR_Ascii	Variable	Read	All

Contains a set of strings containing long unit names corresponding to the enumerations JSR_UnitsEnum.

The string values this property returns can be used with JSR_UnitsEnum as a quick and flexible way to convert the enumeration into human readable text for display.

Although we have tried to think of all possible units for future versions of the JSR Common SDK, it is possible that some future version may have more unit enumerations in JSR_UnitsEnum. If your code is based on always getting this list of unit names from the SDK at run time, your code will handle units added to future versions.

See the section of this document 'Using Enum's and Name Property ID's' for how to use this enum for display purposes.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_String sStringArray[n];
JSR_Ascii sAsciiArray[n];
```

Prototypes

```
JSR_Status JSR_GetString(nHandle, JSR_ID_ReferenceUnitNames, n, sStringArray);
JSR_Status JSR_GetAscii(nHandle, JSR_ID_ReferenceUnitNames, n, sAsciiArray);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_String sStringArray[n];
JSR_Ascii sAsciiArray[n];
```

Prototypes

```
JSR_Status JSRLibFuncs.GetString(nHandle, JSR_ID_ReferenceUnitNames, n, sStringArray);
JSR_Status JSRLibFuncs.GetAscii(nHandle, JSR_ID_ReferenceUnitNames, n, sAsciiArray);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_String sStringArray[n], *pStringArray;
JSR_Ascii sAsciiArray[n], *pAsciiArray;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
// You must allocate buffers before calling
JSR_Status ic.GetArrayString(JSR_ID_ReferenceUnitNames, n, (JSR_String*)sStringArray);
JSR_Status ic.GetArrayString(JSR_ID_ReferenceUnitNames, n, (JSR_Ascii*)sAsciiArray);

// Returns allocated strings that must eventually be freed
JSR_Status ic.GetArrayString(JSR_ID_ReferenceUnitNames, (JSR_String**)&pStringArray);
JSR_Status ic.GetArrayString(JSR_ID_ReferenceUnitNames, (JSR_Ascii**)&pAsciiArray);
```

JSR_Simple_ActiveX.dll

<Not Implemented>

5.4 JSR_ID_ReferenceUnitAbbrNames

Data Type	Count	Attributes	Object
JSR_String or JSR_Ascii	Variable	Read	All

Contains a set of strings containing abbreviated unit names corresponding to the enumerations JSR_UnitsEnum.

This function is the same as JSR_ID_ReferenceUnitNames, except the text returned is the International System of Units (si) abbreviation for the various units. See JSR_ID_ReferenceUnitNames for information on how to use this ID.

Examples of the abbreviations are “cm” vs “centimeter” and “Hz” vs “Hertz.”

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_String sStringArray[n];
JSR_Ascii sAsciiArray[n];
```

Prototypes

```
JSR_Status JSR_GetString(nHandle, JSR_ID_ReferenceUnitAbbrNames, n, sStringArray);
JSR_Status JSR_GetAscii(nHandle, JSR_ID_ReferenceUnitAbbrNames, n, sAsciiArray);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_String sStringArray[n];
JSR_Ascii sAsciiArray[n];
```

Prototypes

```
JSR_Status JSRLibFuncs.GetString(nHandle, JSR_ID_ReferenceUnitAbbrNames, n, sStringArray);
JSR_Status JSRLibFuncs.GetAscii(nHandle, JSR_ID_ReferenceUnitAbbrNames, n, sAsciiArray);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_String sStringArray[n], *pStringArray;
JSR_Ascii sAsciiArray[n], *pAsciiArray;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
// You must allocate buffers before calling
JSR_Status ic.GetArrayString(JSR_ID_ReferenceUnitAbbrNames, n, (JSR_String*)sStringArray);
JSR_Status ic.GetArrayString(JSR_ID_ReferenceUnitAbbrNames, n, (JSR_Ascii*)sAsciiArray);

// Returns allocated strings that must eventually be freed
JSR_Status ic.GetArrayString(JSR_ID_ReferenceUnitAbbrNames, (JSR_String**)&pStringArray);
JSR_Status ic.GetArrayString(JSR_ID_ReferenceUnitAbbrNames, (JSR_Ascii**)&pAsciiArray);
```

JSR_Simple_ActiveX.dll

<Not Implemented>

5.5 JSR_ID_ReferenceObjectTypeNames

Data Type	Count	Attributes	Object
JSR_String or JSR_Ascii	2	Read	All

This property contains a set of strings of object types, which correspond to the JSR_ObjTypeEnum .

Although we have tried to think of all possible units for future versions of the JSR Common SDK, it is possible that some future version may have more unit enumerations in JSR_UnitsEnum. If your code is based on always getting this list of unit names from the SDK at run time, your code will handle units added to future versions.

See the section of this document 'Using Enum's and Name Property ID's' for how to use this enum for display purposes.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_String sStringArray[2];
JSR_Ascii sAsciiArray[2];
```

Prototypes

```
JSR_Status JSR_GetString(nHandle, JSR_ID_ReferenceObjectTypeNames, 2, sStringArray);
JSR_Status JSR_GetAscii(nHandle, JSR_ID_ReferenceObjectTypeNames, 2, sAsciiArray);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_String sStringArray[2];
JSR_Ascii sAsciiArray[2];
```

Prototypes

```
JSR_Status JSRLibFuncs.GetString(nHandle, JSR_ID_ReferenceObjectTypeNames, nCount,
sStringArray);

JSR_Status JSRLibFuncs.GetAscii(nHandle, JSR_ID_ReferenceObjectTypeNames, nCount, sAsciiArray);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_String sStringArray[2], *pStringArray;
JSR_Ascii sAsciiArray[2], *pAsciiArray;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
// You must allocate buffers before calling
JSR_Status ic.GetArrayString(JSR_ID_ReferenceObjectTypeNames, 2, (JSR_String*)sStringArray);
JSR_Status ic.GetArrayString(JSR_ID_ReferenceObjectTypeNames, 2, (JSR_Ascii*)sAsciiArray);

// Returns allocated strings that must eventually be freed
JSR_Status ic.GetArrayString(JSR_ID_ReferenceObjectTypeNames, (JSR_String**)&pStringArray);
JSR_Status ic.GetArrayString(JSR_ID_ReferenceObjectTypeNames, (JSR_Ascii**)&pAsciiArray);
```

JSR_Simple_ActiveX.dll

<Not Implemented>

5.6 JSR_ID_ReferenceEnableNames

Data Type	Count	Attributes	Object
JSR_String or JSR_Ascii	2	Read	All

This property contains a set of two strings (“Enable”/”Disable”) that correspond to the JSR_EnableEnum.

These strings may be used with any property that has zero and one as the only possible values, which includes properties of both types JSR_Bool and JSR_EnableEnum.

Enum	String	Index
JSR_DISABLE	“Disabled”	0
JSR_ENABLE	“Enable”	1

See the section of this document ‘Using Enum’s and Name Property ID’s’ for how to use this enum for display purposes.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_String sStringArray[2];
JSR_Ascii sAsciiArray[2];
```

Prototypes

```
JSR_Status JSR_GetString(nHandle, JSR_ID_ReferenceEnableNames, 2, sStringArray);
JSR_Status JSR_GetAscii(nHandle, JSR_ID_ReferenceEnableNames, 2, sAsciiArray);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_String sStringArray[2];
JSR_Ascii sAsciiArray[2];
```

Prototypes

```
JSR_Status JSRLibFuncs.GetString(nHandle, JSR_ID_ReferenceEnableNames, 2, sStringArray);
JSR_Status JSRLibFuncs.GetAscii(nHandle, JSR_ID_ReferenceEnableNames, 2, sAsciiArray);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_String sStringArray[2], *pStringArray;
JSR_Ascii sAsciiArray[2], *pAsciiArray;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
// You must allocate buffers before calling
JSR_Status ic.GetArrayString(JSR_ID_ReferenceEnableNames, 2, (JSR_String*)sStringArray);
JSR_Status ic.GetArrayString(JSR_ID_ReferenceEnableNames, 2, (JSR_Ascii*)sAsciiArray);

// Returns allocated strings that must eventually be freed
JSR_Status ic.GetArrayString(JSR_ID_ReferenceEnableNames, (JSR_String**)&pStringArray);
JSR_Status ic.GetArrayString(JSR_ID_ReferenceEnableNames, (JSR_Ascii**)&pAsciiArray);
```

JSR_Simple_ActiveX.dll

<Not Implemented>

5.7 JSR_ID_ReferenceTriggerSourceNames

Data Type	Count	Attributes	Object
JSR_String or JSR_Ascii	3	Read	All

This property contains a set of strings that correspond to the JSR_TriggerSourceEnum. This property has three values, although some instruments allow only the first two modes. These names are specifically for use with the ID JSR_ID_ReferenceTriggerSourceNames.

See the section of this document 'Using Enum's and Name Property ID's' for how to use this enum for display purposes.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_String sStringArray[3];
JSR_Ascii sAsciiArray[3];
```

Prototypes

```
JSR_Status JSR_GetString(nHandle, JSR_ID_ReferenceTriggerSourceNames, 3, sStringArray);
JSR_Status JSR_GetAscii(nHandle, JSR_ID_ReferenceTriggerSourceNames, 3, sAsciiArray);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_String sStringArray[3];
JSR_Ascii sAsciiArray[3];
```

Prototypes

```
JSR_Status JSRLibFuncs.GetString(nHandle, JSR_ID_ReferenceTriggerSourceNames, 3, sStringArray);
JSR_Status JSRLibFuncs.GetAscii(nHandle, JSR_ID_ReferenceTriggerSourceNames, 3, sAsciiArray);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_String sStringArray[3], *pStringArray;
JSR_Ascii sAsciiArray[3], *pAsciiArray;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
// You must allocate buffers before calling
JSR_Status ic.GetArrayString(JSR_ID_ReferenceTriggerSourceNames, 3,(JSR_String*)sStringArray);
JSR_Status ic.GetArrayString(JSR_ID_ReferenceTriggerSourceNames, 3,(JSR_Ascii*)sAsciiArray);

// Returns allocated strings that must eventually be freed
JSR_Status ic.GetArrayString(JSR_ID_ReferenceTriggerSourceNames, (JSR_String**)&pStringArray);
JSR_Status ic.GetArrayString(JSR_ID_ReferenceTriggerSourceNames, (JSR_Ascii**)&pAsciiArray);
```

JSR_Simple_ActiveX.dll

<Not Implemented>

5.8 JSR_ID_ReferenceSignalSelectNames

Data Type	Count	Attributes	Object
JSR_String or JSR_Ascii	3	Read	All

This property contains a set of strings that correspond to the JSR_SignalSelectEnum. This property has three values, although some instruments allow only the first two modes. These names are specifically for use with the ID JSR_ID_ReceiverSignalSelect.

See the section of this document 'Using Enum's and Name Property ID's' for how to use this enum for display purposes.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_String sStringArray[3];
JSR_Ascii sAsciiArray[3];
```

Prototypes

```
JSR_Status JSR_GetString(nHandle, JSR_ID_ReferenceSignalSelectNames, 3, sStringArray);
JSR_Status JSR_GetAscii(nHandle, JSR_ID_ReferenceSignalSelectNames, 3, sAsciiArray);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_String sStringArray[3];
JSR_Ascii sAsciiArray[3];
```

Prototypes

```
JSR_Status JSRLibFuncs.GetString(nHandle, JSR_ID_ReferenceSignalSelectNames, 3, sStringArray);
JSR_Status JSRLibFuncs.GetAscii(nHandle, JSR_ID_ReferenceSignalSelectNames, 3, sAsciiArray);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_String sStringArray[3], *pStringArray;
JSR_Ascii sAsciiArray[3], *pAsciiArray;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
// You must allocate buffers before calling
JSR_Status ic.GetArrayString(JSR_ID_ReferenceSignalSelectNames, 3,(JSR_String*)sStringArray);
JSR_Status ic.GetArrayString(JSR_ID_ReferenceSignalSelectNames, 3,(JSR_Ascii*)sAsciiArray);

// Returns allocated strings that must eventually be freed
JSR_Status ic.GetArrayString(JSR_ID_ReferenceSignalSelectNames, (JSR_String**)&pStringArray);
JSR_Status ic.GetArrayString(JSR_ID_ReferenceSignalSelectNames, (JSR_Ascii**)&pAsciiArray);
```

JSR_Simple_ActiveX.dll

<Not Implemented>

5.9 JSR_ID_ReferenceLowHighNames

Data Type	Count	Attributes	Object
JSR_String or JSR_Ascii	2	Read	All

This property contains a set of strings that correspond to no existing enum. These names are specifically for use with the ID JSR_ID_PulserEnergyIndex, for DPR500 remote pulsers that have two energy setting levels. These strings may also be used with any property that has zero and one as the only possible values and the words “low” and “high” are appropriate.

See the section of this document ‘Using Enum’s and Name Property ID’s’ for how to use this enum for display purposes.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_String sStringArray[2];
JSR_Ascii sAsciiArray[2];
```

Prototypes

```
JSR_Status JSR_GetString(nHandle, JSR_ID_ReferenceLowHighNames, 3, sStringArray);
JSR_Status JSR_GetAscii(nHandle, JSR_ID_ReferenceLowHighNames, 3, sAsciiArray);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_String sStringArray[2];
JSR_Ascii sAsciiArray[2];
```

Prototypes

```
JSR_Status JSRLibFuncs.GetString(nHandle, JSR_ID_ReferenceLowHighNames, 3, sStringArray);
JSR_Status JSRLibFuncs.GetAscii(nHandle, JSR_ID_ReferenceLowHighNames, 3, sAsciiArray);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_String sStringArray[2], *pStringArray;
JSR_Ascii sAsciiArray[2], *pAsciiArray;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
// You must allocate buffers before calling
JSR_Status ic.GetArrayString(JSR_ID_ReferenceLowHighNames, 2,(JSR_String*)sStringArray);
JSR_Status ic.GetArrayString(JSR_ID_ReferenceLowHighNames, 2,(JSR_Ascii*)sAsciiArray);

// Returns allocated strings that must eventually be freed
JSR_Status ic.GetArrayString(JSR_ID_ReferenceLowHighNames, (JSR_String**)&pStringArray);
JSR_Status ic.GetArrayString(JSR_ID_ReferenceLowHighNames, (JSR_Ascii**)&pAsciiArray);
```

JSR_Simple_ActiveX.dll

<Not Implemented>

5.10 JSR_ID_ReferenceTriggerEdgeNames

Data Type	Count	Attributes	Object
JSR_String or JSR_Ascii	Variable	Read	All

This property contains a set of strings that correspond to the enum JSR_TriggerEdgeEnum. These names are specifically for use with the property JSR_ID_PulserTriggerEdge.

Although we have tried to think of all possible units for future versions of the JSR Common SDK, it is possible that some future version may have more unit enumerations in JSR_UnitsEnum. If your code is based on always getting this list of unit names from the SDK at run time, your code will handle units added to future versions.

See the section of this document 'Using Enum's and Name Property ID's' for how to use this enum for display purposes.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_String sStringArray[n];
JSR_Ascii sAsciiArray[n];
```

Prototypes

```
JSR_Status JSR_GetString(nHandle, JSR_ID_ReferenceTriggerEdgeNames, n, sStringArray);
JSR_Status JSR_GetAscii(nHandle, JSR_ID_ReferenceTriggerEdgeNames, n, sAsciiArray);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_String sStringArray[n];
JSR_Ascii sAsciiArray[n];
```

Prototypes

```
JSR_Status JSRLibFuncs.GetString(nHandle, JSR_ID_ReferenceTriggerEdgeNames, n, sStringArray);
JSR_Status JSRLibFuncs.GetAscii(nHandle, JSR_ID_ReferenceTriggerEdgeNames, n, sAsciiArray);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_String sStringArray[n], *pStringArray;
JSR_Ascii sAsciiArray[n], *pAsciiArray;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
// You must allocate buffers before calling
JSR_Status ic.GetArrayString(JSR_ID_ReferenceTriggerEdgeNames, n,(JSR_String*)sStringArray);
JSR_Status ic.GetArrayString(JSR_ID_ReferenceTriggerEdgeNames, n,(JSR_Ascii*)sAsciiArray);

// Returns allocated strings that must eventually be freed
JSR_Status ic.GetArrayString(JSR_ID_ReferenceTriggerEdgeNames, (JSR_String**)&pStringArray);
JSR_Status ic.GetArrayString(JSR_ID_ReferenceTriggerEdgeNames, (JSR_Ascii**)&pAsciiArray);
```

JSR_Simple_ActiveX.dll

<Not Implemented>

5.11 JSR_ID_ReferenceOffOnNames

Data Type	Count	Attributes	Object
JSR_String or JSR_Ascii	Variable	Read	All

This property contains a set of two strings that correspond to no enum. These strings may be used with any property that has zero and one as the only possible values, which includes properties of both types JSR_Bool and JSR_EnableEnum.

See the section of this document ‘Using Enum’s and Name Property ID’s’ for how to use this enum for display purposes.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_String sStringArray[n];
JSR_Ascii sAsciiArray[n];
```

Prototypes

```
JSR_Status JSR_GetString(nHandle, JSR_ID_ReferenceOffOnNames, n, sStringArray);
JSR_Status JSR_GetAscii(nHandle, JSR_ID_ReferenceOffOnNames, n, sAsciiArray);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_String sStringArray[n];
JSR_Ascii sAsciiArray[n];
```

Prototypes

```
JSR_Status JSRLibFuncs.GetString(nHandle, JSR_ID_ReferenceOffOnNames, n, sStringArray);
JSR_Status JSRLibFuncs.GetAscii(nHandle, JSR_ID_ReferenceOffOnNames, n, sAsciiArray);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_String sStringArray[n], *pStringArray;
JSR_Ascii sAsciiArray[n], *pAsciiArray;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
// You must allocate buffers before calling
JSR_Status ic.GetArrayString(JSR_ID_ReferenceOffOnNames, n,(JSR_String*)sStringArray);
JSR_Status ic.GetArrayString(JSR_ID_ReferenceOffOnNames, n,(JSR_Ascii*)sAsciiArray);

// Returns allocated strings that must eventually be freed
JSR_Status ic.GetArrayString(JSR_ID_ReferenceOffOnNames, (JSR_String**)&pStringArray);
JSR_Status ic.GetArrayString(JSR_ID_ReferenceOffOnNames, (JSR_Ascii**)&pAsciiArray);
```

JSR_Simple_ActiveX.dll

<Not Implemented>

5.12 JSR_ID_ReferenceFalseTrueNames

Data Type	Count	Attributes	Object
JSR_String or JSR_Ascii	Variable	Read	All

This property contains a set of two strings, “False” and “True”. These strings may be used with any property that has zero and one as the only possible values, especially JSR_Bool.

Enum	String	Index
JSR_FALSE	“False”	0
JSR_TRUE	“True”	1

See the section of this document ‘Using Enum’s and Name Property ID’s’ for how to use this enum for display purposes.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_String sStringArray[2];
JSR_Ascii sAsciiArray[2];
```

Prototypes

```
JSR_Status JSR_GetString(nHandle, JSR_ID_ReferenceFalseTrueNames, 3, sStringArray);
JSR_Status JSR_GetAscii(nHandle, JSR_ID_ReferenceFalseTrueNames, 3, sAsciiArray);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_String sStringArray[2];
JSR_Ascii sAsciiArray[2];
```

Prototypes

```
JSR_Status JSRLibFuncs.GetString(nHandle, JSR_ID_ReferenceFalseTrueNames, 3, sStringArray);
JSR_Status JSRLibFuncs.GetAscii(nHandle, JSR_ID_ReferenceFalseTrueNames, 3, sAsciiArray);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_String sStringArray[2], *pStringArray;
JSR_Ascii sAsciiArray[2], *pAsciiArray;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
// You must allocate buffers before calling
JSR_Status ic.GetArrayString(JSR_ID_ReferenceFalseTrueNames, 2,(JSR_String*)sStringArray);
JSR_Status ic.GetArrayString(JSR_ID_ReferenceFalseTrueNames, 2,(JSR_Ascii*)sAsciiArray);

// Returns allocated strings that must eventually be freed
JSR_Status ic.GetArrayString(JSR_ID_ReferenceFalseTrueNames, (JSR_String**)&pStringArray);
JSR_Status ic.GetArrayString(JSR_ID_ReferenceFalseTrueNames, (JSR_Ascii**)&pAsciiArray);
```

JSR_Simple_ActiveX.dll

<Not Implemented>

6 Baseline Properties of the Library Object

The library object is the cornerstone of the JSR Library. Due to that, the Library has a large collection of properties that are used to get basic information about the version of the JSR Library, and other version specific resources. Like all objects the Library Property has all the Standard Properties, and a selection of Object Specific Properties. By definition, the Library has no Extended Properties.

6.1 JSR_ID_LibraryName

Data Type	Count	Attributes	Object
JSR_String or JSR_Ascii	1	Read	Library

Returns a text string identifying the Library by name. For versions of the library that are in the series 1.X, that string is" JSR Common API Library".

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_String sString;
JSR_Ascii sAscii;
```

Prototypes

```
JSR_Status JSR_GetString(nHandle, JSR_ID_LibraryName, 1, sString);
JSR_Status JSR_GetAscii(nHandle, JSR_ID_LibraryName, 1, sAscii);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_String sString;
JSR_Ascii sAscii;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetString(nHandle, JSR_ID_LibraryName, 1, sString);
JSR_Status JSRLibFuncs.GetAscii(nHandle, JSR_ID_LibraryName, 1, sAscii);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_String sString;
JSR_Ascii sAscii;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetString(JSR_ID_LibraryName, &sString);
JSR_Status ic.GetString(JSR_ID_LibraryName, &sAscii);
```

JSR_Simple_ActiveX.dll

Declarations

```
Dim ic as JSR_Simple
Dim Name as String
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
Name = ic.PropertyString(JSR_ID_LibraryName)
```

6.2 JSR_ID_LibraryVersion

Data Type	Count	Attributes	Object
JSR_Int32	4	Read	Library
JSR_String or JSR_Ascii (JSR_Simple)	1		

Contains a 4-element Int32 array of the version number for the instance of the JSR Library currently running. The elements are in the standard Major, Minor, Revision, Patch order. Library versions with the same Major and Minor Version number should have no substantial functionality changes. The JSR_String and JSR_Ascii forms of the JSR_Simple interface return strings in the form of "*Major*.*Minor*.*Revision*.*Patch*"

Changes to Major or Minor version indicate a change of functionality or other substantial changes, and the Release Notes documentation should be reviewed before upgrading.

This version number exactly matches the JSR_Common.dll's resource version information.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Int32 nVer[4];
```

Prototypes

```
JSR_Status JSR_GetInt32(nHandle, JSR_ID_LibraryVersion, 4, nVer);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Int32 never[4];
```

Prototypes

```
JSR_Status JSRLibFuncs.GetInt32(nHandle, JSR_ID_LibraryName, 4, nVer);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Int32 nVer[4];
JSR_String sString;
JSR_Ascii sAscii;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
// Get 4 element integer version
JSR_Status ic.GetArrayInt32(JSR_ID_LibraryName, 4, nVer);

// Get version as string
JSR_Status ic.GetString(JSR_ID_LibraryName, &sString);
JSR_Status ic.GetString(JSR_ID_LibraryName, &sAscii);
```

JSR_Simple_ActiveX.dll

Declarations

```
Dim ic as JSR_Simple
Dim Version as String
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
Version = ic.PropertyString(JSR_ID_LibraryVersion)
```

6.3 JSR_ID_LibrarySupportedModels

Data Type	Count	Attributes	Object
JSR_Bool	Variable	Read	Library

Contains an array of JSR_Bool values correspond to the JSR_ModelEnum.

These values indicate all the models supported by this version of the JSR_Common.DLL, even if the underlying drivers are not installed, regardless of the arguments to JSR_OpenLibrary(…). If a driver is not installed, that model may be listed here, but that particular model will not work.

To see which models are available and loaded properly for use, see the ID JSR_ID_LibraryDriversStatus.

For forward compatibility via 'DLL replacement only', be sure to not hard-code the size of the supported Models array, but instead use JSR_GetInfo() to get the elementCount before allocating an array for the models list.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Bool bArray[n];
```

Prototypes

```
JSR_Status JSR_GetBool(nHandle, JSR_ID_LibrarySupportedModels, n, bArray);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Bool bArray[n];
```

Prototypes

```
JSR_Status JSRLibFuncs.GetBool(nHandle, JSR_ID_LibrarySupportedModels, n, bArray);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Bool bArray[n];
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetArrayBool(JSR_ID_LibrarySupportedModels, n, bArray);
```

JSR_Simple_ActiveX.dll

<Not Implemented>

6.4 JSR_ID_LibraryDriversStatus

Data Type	Count	Attributes	Object
JSR_Status	Variable	Read	Library

Contains an array of JSR_Status values, which correspond to the JSR_ModelEnum. These values indicate the current driver status for each instrument type.

These values are set during the call to JSR_OpenLibrary(), and if a driver is not properly loaded at that time, there is no way to load the driver without closing and reopening the library. This ID should be checked after if the call to JSR_OpenLibrary() returned any error, especially in the case of an error code JSR_WARN_NOT_ALL_DRIVERS_FOUND.

The JSR_Status values in the array correspond to index values of JSR_MODEL_UNKNOWN and JSR_MODEL_ANY will always be JSR_FAIL_MODEL_INDEX_NOT_USED and should be ignored.

For forward compatibility via 'DLL replacement only', be sure to not hard-code the size of the supported status array, but instead use JSR_GetInfo() to get the elementCount before allocating an array for the status list.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Status nArray[n];
```

Prototypes

```
JSR_Status JSR_GetInt32(nHandle, JSR_ID_LibraryDriverStatus, n, (JSR_Int32*)nArray);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Status nArray[n];
```

Prototypes

```
JSR_Status JSRLibFuncs.GetInt32(nHandle, JSR_ID_LibraryDriverStatus, n, (JSR_Int32*)nArray);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Status nArray[n];
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetArrayInt32(JSR_ID_LibraryDriverStatus, n, (JSR_Int32*)nArray);
```

JSR_Simple_ActiveX.dll

<Not Implemented>

6.5 JSR_ID_LibraryInstrumentHandles

Data Type	Count	Attributes	Object
JSR_Handle	Variable	Read	Library

Contains a list of all instrument objects found by the Library. Which instrument models are searched for depends on the arguments to JSR_OpenLibrary(…). Which instruments are found is also dependent on the proper installation of DLL's and/or drivers that operate with the JSR_Common.DLL to support specific instruments. If there is an unexpected error, or missing ID's in this list, read JSR_ID_LibraryDriversStatus and JSR_ID_LibrarySupportedModels for debugging information.

Instruments in this list are instruments that have been 'found' and are not guaranteed to have loaded properly. If there is/was some form of load error for an instrument with a handle in the list, that error code will be returned by the call to JSR_OpenObject() for that instrument handle. When debugging, be aware that (due to library design) that error may have happened prior to the JSR_OpenObject() call.

Since the list of Instruments is variable based on several options/parameters/settings, be sure to always use JSR_GetInfo() to get the list elementCount before allocating an array for the list.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Handle nArray[n];
```

Prototypes

```
JSR_Status JSR_GetInt32(nHandle, JSR_ID_LibraryInstrumentHandles, n, (JSR_Int32*)nArray);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Handle nArray[n];
```

Prototypes

```
JSR_Status JSRLibFuncs.GetInt32(nHandle, JSR_ID_LibraryInstrumentHandles,n, (JSR_Int32*)nArray);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Handle nArray[n];
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetArrayInt32(JSR_ID_LibraryInstrumentHandles, n, (JSR_Int32*)nArray);
```

JSR_Simple_ActiveX.dll

<Not Implemented>

7 Baseline Properties of an Instrument Object

Every Object (as explained in the Property Overview) has its own Property ID range, consisting of Baseline and Extended Properties. Every Object of a given object type has every baseline property. This section is a list of all Instrument Baseline Properties, their values, settings, and information on how they use each property.

All instrument objects also contain the Standard Properties and can access the Reference Properties.

7.1 JSR_ID_InstrumentChannelHandles

Data Type	Count	Attributes	Object
JSR_Handle	Variable	Read	Instrument

Returns a list of the handles to channel objects within the instrument.

If there was some form of error when the DLL attempted to connect to a specific channel, the handle for the channel will be in this list, but the call to JSR_OpenObject() for that channel will return an error code indicating what error was found when connecting to the channel.

PRC50 and DPR300:

These instruments have only one channel, which is always channel “A”.

DPR500 Only:

These instruments may be purchased configured to have either one channel (which will always be “A”) or two channels, (“A” and “B”). This property reports the count and their handle values.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Handle nArray[n];
```

Prototypes

```
JSR_Status JSR_GetInt32(nHandle, JSR_ID_InstrumentChannelHandles, n, (JSR_Int32*)nArray);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Handle nArray[n];
```

Prototypes

```
JSR_Status JSRLibFuncs.GetInt32(nHandle, JSR_ID_InstrumentChannelHandles,n, (JSR_Int32*)nArray);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Handle nArray[n];
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetArrayInt32(JSR_ID_InstrumentChannelHandles, n, (JSR_Int32*)nArray);
```

JSR_Simple_ActiveX.dll

<Not Implemented>

7.2 JSR_ID_InstrumentModelName

Data Type	Count	Attributes	Object
JSR_String or JSR_Ascii	1	Read	Instrument

Contains the model string of an instrument, such as “PRC50” or “DPR500”. The string returned by this property is normally the same as the model name in JSR_ID_ReferenceModelNames but they may not always be exactly the same.

JSR makes custom instruments based on standard products, which may have a different name than the standard product, usually with a custom prefix or suffix such as “DPR500B” . We suggest host application code should not compare this string from this property to determine features available. Instead, use the JSR_ID_InstrumentModelEnum integer property described below.

When the Library is running in simulation mode, the model name is prefixed with “Sim “. E.g. “Sim PRC50”

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_String sString;
JSR_Ascii sAscii;
```

Prototypes

```
JSR_Status JSR_GetString(nHandle, JSR_ID_InstrumentModelName, 1, sString);
JSR_Status JSR_GetAscii(nHandle, JSR_ID_InstrumentModelName, 1, sAscii);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_String sString;
JSR_Ascii sAscii;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetString(nHandle, JSR_ID_InstrumentModelName, 1, sString);
JSR_Status JSRLibFuncs.GetAscii(nHandle, JSR_ID_InstrumentModelName, 1, sAscii);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_String sString;
JSR_Ascii sAscii;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetString(JSR_ID_InstrumentModelName, sString);
JSR_Status ic.GetString(JSR_ID_InstrumentModelName, sAscii);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM Name as String
```

Specific Prototype

<Not Implemented>

Generic Prototype

```
Name = ic.PropertyString(JSR_ID_InstrumentModelName)
```

7.3 JSR_ID_InstrumentModelEnum

Data Type	Count	Attributes	Object
JSR_ModelEnum	1	Read	Instrument

Returns the enum value that identifies the instrument, that corresponds to the values in JSR_ModelEnum. See also JSR_ID_ReferenceModelNames.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_ModelEnum nModel;
```

Prototypes

```
JSR_Status JSR_GetInt32(nHandle, JSR_ID_InstrumentModelEnum, 1, (JSR_Int32*)&nModel);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_ModelEnum nModel;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetInt32(nHandle, JSR_ID_InstrumentModelEnum, 1 ,(JSR_Int32*)&nModel);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_ModelEnum nModel;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetInt32(JSR_ID_InstrumentModelEnum, (JSR_Int32*)&nModel);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM nModel as JSR_ModelEnum
```

Specific Prototype

<Not Implemented>

Generic Prototype

```
nModel = ic.PropertyInt32(JSR_ID_InstrumentModelEnum)
```

7.4 JSR_ID_InstrumentSerNum

Data Type	Count	Attributes	Object
JSR_String or JSR_Ascii	1	Read	Instrument

Contains the serial number string of the JSR instrument. This is the same serial number for the instrument as a whole, and matches the serial number visible on the product identification tag on the outside of the product. The serial number format may be different for different product lines.

This serial number is usually shown on the outside of the product packaging as well, and will show on shipping orders. This is the serial number you would use to when you communicate with JSR with questions about one specific instrument.

When the Library is running in simulation mode, the returned value will be "Sim Product SN".

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_String sString;
JSR_Ascii sAscii;
```

Prototypes

```
JSR_Status JSR_GetString(nHandle, JSR_ID_InstrumentSerNum e, 1, sString);
JSR_Status JSR_GetAscii(nHandle, JSR_ID_InstrumentSerNum, 1, sAscii);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_String sString;
JSR_Ascii sAscii;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetString(nHandle, JSR_ID_InstrumentSerNum, 1, sString);
JSR_Status JSRLibFuncs.GetAscii(nHandle, JSR_ID_InstrumentSerNum, 1, sAscii);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_String sString;
JSR_Ascii sAscii;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetString(JSR_ID_InstrumentSerNum, sString);
JSR_Status ic.GetString(JSR_ID_InstrumentSerNum, sAscii);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
```
DIM Name as String

Specific Prototype

<Not Implemented>

Generic Prototype

```
Name = ic.PropertyString(JSR_ID_InstrumentSerNum)
```

7.5 JSR_ID_InstrumentConnectStatus

Data Type	Count	Attributes	Object
JSR_Status	1	Read	Instrument

This contains the current connection status of the instrument. The return JSR_Status value of the read function itself reports success of the read function, and the 'data' of the read function contains the Instrument connection status.

For instruments connected via a cable, this property will cause an "acknowledge" message to be sent to the instrument and will wait for a reply before returning. If the reply takes too long, the instrument has been disconnected, or it has its power off, this property will return a fail status. A read of this property may take up to 500 ms while waiting for a timeout.

Error codes specific to this ID are :

JSR_FAIL_INSTRUMENT_POWER_DOWN_THEN_UP,
JSR_FAIL_INSTRUMENT_DISCONNECTED
JSR_FAIL_DPR_COMMO_FAILURE

Value JSR_FAIL_INSTRUMENT_POWER_DOWN_THEN_UP indicates power to the instrument was turned off, then back on since the last command was sent to the instrument. You host application may need to send all of the settings back to the instrument in this case. This error condition will get cleared after the first new value is successfully sent to the instrument.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Status nStatus;
```

Prototypes

```
JSR_Status JSR_GetInt32(nHandle, JSR_ID_InstrumentConnectStatus, 1, (JSR_Int32*)& nStatus);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_ModelEnum nModel;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetInt32(nHandle, JSR_ID_InstrumentConnectStatus,
1,(JSR_Int32*)&nStatus);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_ModelEnum nModel;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetInt32(JSR_ID_InstrumentConnectStatus, (JSR_Int32*)&nStatus);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM nStatus as JSR_Status
```

Specific Prototype

<Not Implemented>

Generic Prototype

```
nStatus = ic.PropertyStatus(JSR_ID_InstrumentConnectStatus)
```

7.6 JSR_ID_InstrumentHasManualControls

Data Type	Count	Attributes	Object
JSR_Bool	1	Read	Instrument

The returned value will be JSR_TRUE if the JSR instrument has manual controls on its front panel, otherwise JSR_FALSE.

Some DPR300's have manual controls, and will return JSR_TRUE, since no other JSR instrument has manual controls, all other instruments will return JSR_FALSE.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Bool bState;
```

Prototypes

```
JSR_Status JSR_GetBool(nHandle, JSR_ID_InstrumentHasManualControls, 1, &bState);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Bool bState;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetBool(nHandle, JSR_ID_InstrumentHasManualControls,
1,(JSR_Int32*)&bState);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Bool bState;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetBool(JSR_ID_InstrumentHasManualControls, &bState);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM bState as Bool
```

Specific Prototype

<Not Implemented>

Generic Prototype

```
bState = ic.PropertyBool(JSR_ID_InstrumentHasManualControls)
```

8 Extended Properties of an Instrument Object

The set of extended Properties of any particular instrument model will be a subset of these properties. If an instrument does not have one of these properties, the status return of an attempt to access the property will be JSR_FAIL_ID_NOT_VALID.

8.1 JSR_ID_InstrumentPCISlot

Data Type	Count	Attributes	Object
JSR_Int32	1	Read	Instrument (PRC50)

PRC50 Only:

If supported by the mother board hardware, the mother board BIOS, and mother board drivers, this property contains the number for the PCI slot containing the PRC50.

There is not an industry-wide standard, requirement, or even a defacto standard, so this property may not function on all platforms. Note that this property reports the Physical PCI slot, which is not the same as the PCI bus number or PCI device number within a PCI bus group.

On all platforms that provide the PCI slot number function, the first PCI slot is 1. When the JSR Common API determines the information is not present or gets an error from the operating system when attempting to read the slot number, the API will return minus one or zero as the slot number. The JSR Common API will not return an error status value.

This property is called internally by the JSR Common API in order to create the string returned by the property JSR_ID_ChannelDescription.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Int32 nSlot;
```

Prototypes

```
JSR_Status JSR_GetInt32(nHandle, JSR_ID_InstrumentPCISlot, 1, (JSR_Int32*)&nSlot);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Int32 nSlot;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetInt32(nHandle, JSR_ID_InstrumentPCISlot, 1,(JSR_Int32*)&nSlot);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Int32 nSlot;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetInt32(JSR_ID_InstrumentPCISlot, (JSR_Int32*)&nSlot);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM nSlot as Integer
```

Specific Prototype

<Not Implemented>

Generic Prototype

```
nSlot = ic.PropertyInt32(JSR_ID_InstrumentPCISlot)
```

8.2 JSR_ID_InstrumentFirmwareVer

Data Type	Count	Attributes	Object
JSR_String or JSR_Ascii	1	Read	Instrument (PRC50)

PRC50 Only:

Contains the firmware version string of the JSR PRC50 instrument.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_String sString;
JSR_Ascii sAscii;
```

Prototypes

```
JSR_Status JSR_GetString(nHandle, JSR_ID_InstrumentFirmwareVer e, 1, sString);
JSR_Status JSR_GetAscii(nHandle, JSR_ID_InstrumentFirmwareVer, 1, sAscii);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_String sString;
JSR_Ascii sAscii;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetString(nHandle, JSR_ID_InstrumentFirmwareVer, 1, sString);
JSR_Status JSRLibFuncs.GetAscii(nHandle, JSR_ID_InstrumentFirmwareVer, 1, sAscii);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_String sString;
JSR_Ascii sAscii;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetString(JSR_ID_InstrumentFirmwareVer, sString);
JSR_Status ic.GetString(JSR_ID_InstrumentFirmwareVer, sAscii);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
```
DIM Version as String

Specific Prototype

<Not Implemented>

Generic Prototype

```
Version = ic.PropertyString(JSR_ID_InstrumentFirmwareVer)
```

8.3 JSR_ID_InstrumentSerialComPort

Data Type	Count	Attributes	Object
JSR_Int32	1	Read	Instrument (Extended)

DPR500 & DPR300:

Returns the number of the COM port an RS-232 instrument is connected to. The first possible number is one since there is no COM0.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Int32 nPort;
```

Prototypes

```
JSR_Status JSR_GetInt32(nHandle, JSR_ID_InstrumentPCISlot, 1, (JSR_Int32*)&nPort);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Int32 nPort;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetInt32(nHandle, JSR_ID_InstrumentPCISlot, 1,(JSR_Int32*)&nPort);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Int32 nPort;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetInt32(JSR_ID_InstrumentPCISlot, (JSR_Int32*)&nPort);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM nPort as integer
```

Specific Prototype

<Not Implemented>

Generic Prototype

```
nPort = ic.PropertyInt32(JSR_ID_InstrumentPCISlot)
```

8.4 JSR_ID_InstrumentSerialChainAddress

Data Type	Count	Attributes	Object
JSR_Int32	1	Read	Instrument (Extended)

DPR500 & DPR300:

These instruments allow “pass through” mode to daisy-chain up to 255 instruments on a single RS-232 serial port by assigning a Cahin Address to each instrument during DLL initialization. This property reports the address. The first instrument on the serial port has address 1.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Int32 nAddress;
```

Prototypes

```
JSR_Status JSR_GetInt32(nHandle, JSR_ID_InstrumentSerialChainAddress, 1, &nPort);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Int32 nAddress;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetInt32(nHandle, JSR_ID_InstrumentSerialChainAddress, 1, &nAddress);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Int32 nAddress;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetInt32(JSR_ID_InstrumentSerialChainAddress, &nAddress);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM nAddress as Integer
```

Specific Prototype

<Not Implemented>

Generic Prototype

```
nAddress = ic.PropertyInt32(JSR_ID_InstrumentSerialChainAddress)
```

8.5 JSR_ID_InstrumentPowerLEDControl

Data Type	Count	Attributes	Object
JSR_Int32	1	Read-Write	Instrument (Extended)

DPR500 & DPR300:
Contains the state of the “Power” indicator LED on instruments that have a “power” LED. The allowed values are 0 to 255, where the blink rate increases from slow to fast.
0 - blink slowly 254 - blink fast 255 - steady on

Due to rounding, the value read back from this property may be off by one from the value your application set to this property.

PRC50:
The PRC50 does not have a Power LED and therefore does not have this property.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Int32 nVal;
```

Prototypes

```
JSR_Status JSR_GetInt32(nHandle, JSR_ID_InstrumentPowerLEDControl, 1, (JSR_Int32*)&nVal);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Int32 nVal;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetInt32(nHandle, JSR_ID_InstrumentPowerLEDControl, 1,(JSR_Int32*)&nVal);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Int32 nVal;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetInt32(JSR_ID_InstrumentPowerLEDControl, (JSR_Int32*)&nVal);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM nVal as Integer
```

Specific Prototype

<Not Implemented>

Generic Prototype

```
nVal = ic.PropertyInt32(JSR_ID_InstrumentPowerLEDControl)
```

9 Baseline Properties of a Channel Object

Every channel object for every make and model of instrument will have every channel baseline property.

On instruments that have only one channel, that channel will always be defined as channel “A”.

All channel objects also contain the Standard Properties and can access the Reference Properties.

9.1 JSR_ID_ChannelDescription

Data Type	Count	Attributes	Object
JSR_String or JSR_Ascii	1	Read	Channel

This property contains a string with the combination of the channel's instrument model, identification, and address. It is primarily for display on a graphical user interface. The format varies between instrument models to adapt to the different instruments.

PRC50 Examples:

"PRC50, PCI Slot 2, 50 MHz" or "PRC50, PCI Slot 1, 50 MHz"

The property JSR_ID_InstrumentPCISlot is called internally by the JSR Common API to determine the slot number. See the description for JSR_ID_InstrumentPCISlot about why this property may report zero on some platforms.

DPR500 Examples:

"DPR500, COM4, Addr 2, Ch B, 300 MHz, RP-H1"
"DPR500, COM1, Addr 2, Ch B, 300 MHz, No Pulser"

DPR300 Example:

"DPR300, COM3, Addr 1, 50 MHz"

When the Library is operated in Simulate mode, the prefix "Sim " will be placed in front of the rest of the string described above.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_String sString;
JSR_Ascii sAscii;
```

Prototypes

```
JSR_Status JSR_GetString(nHandle, JSR_ID_ChannelDescription, 1, sString);
JSR_Status JSR_GetAscii(nHandle, JSR_ID_ChannelDescription, 1, sAscii);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_String sString;
JSR_Ascii sAscii;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetString(nHandle, JSR_ID_ChannelDescription, 1, sString);
JSR_Status JSRLibFuncs.GetAscii(nHandle, JSR_ID_ChannelDescription, 1, sAscii);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_String sString;
JSR_Ascii sAscii;
```

Specific Prototypes

```
JSR_Status GetChannelDescriptionString(sString);
JSR_Status GetChannelDescriptionString(sAscii);
```

Generic Prototypes

```
JSR_Status ic.GetString(JSR_ID_ChannelDescription, sString);
JSR_Status ic.GetString(JSR_ID_ChannelDescription, sAscii);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM Desc as String
```

Specific Prototype

```
Desc = ic.ChannelDescription
```

Generic Prototype

```
Desc = ic.PropertyString(JSR_ID_ChannelDescription)
```

9.2 JSR_ID_ChannelLetter

Data Type	Count	Attributes	Object
JSR_String or JSR_Ascii	1	Read	Channel

Contains a string of a single character, which will always be either uppercase “A” or uppercase “B”.

PRC50 and DPR300 :

These instruments can have only one channel, which is always “A”

DPR500:

This instrument can have one or two channels, “A” or “B”.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_String sString;
JSR_Ascii sAscii;
```

Prototypes

```
JSR_Status JSR_GetString(nHandle, JSR_ID_ChannelLetter, 1, sString);
JSR_Status JSR_GetAscii(nHandle, JSR_ID_ChannelLetter, 1, sAscii);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_String sString;
JSR_Ascii sAscii;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetString(nHandle, JSR_ID_ChannelLetter, 1, sString);
JSR_Status JSRLibFuncs.GetAscii(nHandle, JSR_ID_ChannelLetter, 1, sAscii);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_String sString;
JSR_Ascii sAscii;
```

Specific Prototypes

```
JSR_Status GetChannelLetter(sString);
JSR_Status GetChannelLetter(sAscii);
```

Generic Prototypes

```
JSR_Status ic.GetString(JSR_ID_ChannelLetter, sString);
JSR_Status ic.GetString(JSR_ID_ChannelLetter, sAscii);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM Letter as String
```

Specific Prototype

```
Letter = ic.ChannelLetter
```

Generic Prototype

```
Letter = ic.PropertyString(JSR_ID_ChannelLetter)
```

9.3 JSR_ID_ChannelPulserHandles

Data Type	Count	Attributes	Object
JSR_Handle	1 or 0 - 16	Read	Channel

This property contains the handle(s) for pulser(s) connected to the specified channel handle.

PRC50 and DPR300:

These instruments have only one pulser per channel. A “Get” of the JSR_ID_ChannelPulserHandles property will therefore always provide one pulser handle.

DPR500:

This instrument can have from 1 to 16 pulsers connected through a MUX1600 multiplier.

Some JSR instruments allow a hardware configuration where a channel does not contain a pulser. In this case the elementCount and byteCount fields of the info struct for this property will contain the value zero. In this case a call to JSR_GetInt32 using this ID will return the status of JSR_FAIL_NO_INSTRUMENT_FOUND.

If there was an error when the DLL attempted to connect to a specific pulser, the handle for the pulser will be in this list, but the call to JSR_OpenObject() for that pulser will return an error code indicating the type of error.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Handle nArray[n];
```

Prototypes

```
JSR_Status JSR_GetInt32(nHandle, JSR_ID_ChannelPulserHandles, n, (JSR_Int32*)nArray);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Handle nArray[n];
```

Prototypes

```
JSR_Status JSRLibFuncs.GetInt32(nHandle, JSR_ID_ChannelPulserHandles,n, (JSR_Int32*)nArray);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Handle nArray[n];
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetArrayInt32(JSR_ID_ChannelPulserHandles, n, (JSR_Int32*)nArray);
```

JSR_Simple_ActiveX.dll

<Not Implemented>

9.4 JSR_ID_ChannelReceiverHandles

Data Type	Count	Attributes	Object
JSR_Handle	1	Read	Channel

This property contains the handle for the receiver connected to the specified channel handle.

Currently, all JSR instruments have only one receiver per channel, however future products may have more than one receiver per channel.

Some instruments allow a hardware configuration where a channel does not contain a receiver. In this case the elementCount and byteCount fields of the info struct for this property will contain the value zero. A call to JSR_GetInt32() using this ID will return the status value JSR_FAIL_NO_INSTRUMENT_FOUND.

If there was an error when the DLL attempted to connect to a specific receiver, the handle for the receiver will be in this list, but the call to JSR_OpenObject() for that receiver will return an error code indicating the type of error.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Handle nArray[n];
```

Prototypes

```
JSR_Status JSR_GetInt32(nHandle, JSR_ID_ChannelReceiverHandles, n, (JSR_Int32*)nArray);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Handle nArray[n];
```

Prototypes

```
JSR_Status JSRLibFuncs.GetInt32(nHandle, JSR_ID_ChannelReceiverHandles,n, (JSR_Int32*)nArray);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Handle nArray[n];
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetArrayInt32(JSR_ID_ChannelReceiverHandles, n, (JSR_Int32*)nArray);
```

JSR_Simple_ActiveX.dll

<Not Implemented>

10 Extended Properties of a Channel Object

There are no extended properties for channel objects at this time.

11 Baseline Properties of a Pulser Object

Every pulser object for every instrument make and model will have these baseline properties.

All pulser objects also contain the Standard Properties and can access the Reference Properties.

11.1 JSR_ID_PulserModelName

Data Type	Count	Attributes	Object
JSR_String or JSR_Ascii	1	Read	Pulser

This property contains a string value that identifies the pulser model. Many instruments have several pulser units which are interchangeable, so that the ModelName changes based on which remote pulser was plugged in at the time the Library was opened.

Example Strings: “PRC50” “RP-4H” “RP-4U” “DPR300”

When the Library is operated in Simulate mode, the string will be prefixed with “Sim “

The displayText field of JSR_InfoStruct and JSR_AsciiInfoStruct for this property is always “Model”.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_String sString;
JSR_Ascii sAscii;
```

Prototypes

```
JSR_Status JSR_GetString(nHandle, JSR_ID_PulserModelName, 1, sString);
JSR_Status JSR_GetAscii(nHandle, JSR_ID_PulserModelName, 1, sAscii);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_String sString;
JSR_Ascii sAscii;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetString(nHandle, JSR_ID_PulserModelName, 1, sString);
JSR_Status JSRLibFuncs.GetAscii(nHandle, JSR_ID_PulserModelName, 1, sAscii);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_String sString;
JSR_Ascii sAscii;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetString(JSR_ID_PulserModelName, sString);
JSR_Status ic.GetString(JSR_ID_PulserModelName, sAscii);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM Name as String
```

Specific Prototype

<Not Implemented>

Generic Prototype

```
Name = ic.PropertyString(JSR_ID_PulserModelName)
```

11.2 JSR_ID_PulserSerNum

Data Type	Count	Attributes	Object
JSR_String or JSR_Ascii	1	Read	Pulser

This property contains a string value which reports:

PRC50	serial number of the main board, e.g. “MB0001”
DPR500	serial number of the remote pulser, e.g. “GB0124”

The “displayText” element of JSR_InfoStruct and JSR_AsciiInfoStruct for this property may be different for different instruments:

Model	JSR_InfoStruct.displayText	String when in simulation mode
PRC50	“Main Board SN”	“Sim Pulser SN”
DPR300	“Serial Number”	“Sim Pulser SN”
DPR500	“Serial Number”	“Sim Ch A Pulser SN”

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_String sString;
JSR_Ascii sAscii;
```

Prototypes

```
JSR_Status JSR_GetString(nHandle, JSR_ID_PulserSerNum, 1, sString);
JSR_Status JSR_GetAscii(nHandle, JSR_ID_PulserSerNum, 1, sAscii);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_String sString;
JSR_Ascii sAscii;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetString(nHandle, JSR_ID_PulserSerNum, 1, sString);
JSR_Status JSRLibFuncs.GetAscii(nHandle, JSR_ID_PulserSerNum, 1, sAscii);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_String sString;
JSR_Ascii sAscii;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetString(JSR_ID_PulserSerNum, sString);
JSR_Status ic.GetString(JSR_ID_PulserSerNum, sAscii);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM Serial as String
```

Specific Prototype

<Not Implemented>

Generic Prototype

```
Serial = ic.PropertyString(JSR_ID_PulserSerNum)
```

11.3 JSR_ID_PulserHardwareSerNum

Data Type	Count	Attributes	Object
JSR_String or JSR_Ascii	1	Read	Pulser

This property contains a string value which reports:

PRC50	the string “Not Used”
DPR300	The string “Not Used”
DPR500	hardware serial number of the remote pulser, e.g. “31CC3”

The “displayText” element of JSR_InfoStruct and JSR_AsciiInfoStruct for this property may be different for different instruments:

Model	JSR_InfoStruct.displayText	String when in simulation mode
PRC50	“Hardware Serial Number”	“Sim Pulser HW SN”
DPR300	“Not Used”	“Sim Pulser HW SN”
DPR500	“Hardware Serial Number”	“Sim Ch A Pulser HW SN”

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_String sString;
JSR_Ascii sAscii;
```

Prototypes

```
JSR_Status JSR_GetString(nHandle, JSR_ID_PulserHardwareSerNum, 1, sString);
JSR_Status JSR_GetAscii(nHandle, JSR_ID_PulserHardwareSerNum, 1, sAscii);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_String sString;
JSR_Ascii sAscii;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetString(nHandle, JSR_ID_PulserHardwareSerNum, 1, sString);
JSR_Status JSRLibFuncs.GetAscii(nHandle, JSR_ID_PulserHardwareSerNum, 1, sAscii);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_String sString;
JSR_Ascii sAscii;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetString(JSR_ID_PulserHardwareSerNum, sString);
JSR_Status ic.GetString(JSR_ID_PulserHardwareSerNum, sAscii);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM Serial as String
```

Specific Prototype

<Not Implemented>

Generic Prototype

```
Serial = ic.PropertyString(JSR_ID_PulserHardwareSerNum)
```

11.4 JSR_ID_PulserHardwareRev

Data Type	Count	Attributes	Object
JSR_String or JSR_Ascii	1	Read	Instrument

String value that identifies the hardware revision.

Examples: “A”, “B”, “C”

The “displayText” element of JSR_InfoStruct and JSR_AsciiInfoStruct for this property may be different for different instruments:

Model	JSR_InfoStruct.displayText	String when in simulation mode
PRC50	“Main Board Rev”	“Sim Pulser Rev”
DPR300	“Revision”	“Sim Pulser Rev”
DPR500	“Revision”	“Sim Ch A Pulser Rev”

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_String sString;
JSR_Ascii sAscii;
```

Prototypes

```
JSR_Status JSR_GetString(nHandle, JSR_ID_PulserHardwareSerNum, 1, sString);
JSR_Status JSR_GetAscii(nHandle, JSR_ID_PulserHardwareSerNum, 1, sAscii);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_String sString;
JSR_Ascii sAscii;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetString(nHandle, JSR_ID_PulserHardwareSerNum, 1, sString);
JSR_Status JSRLibFuncs.GetAscii(nHandle, JSR_ID_PulserHardwareSerNum, 1, sAscii);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_String sString;
JSR_Ascii sAscii;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetString(JSR_ID_PulserHardwareSerNum, sString);
JSR_Status ic.GetString(JSR_ID_PulserHardwareSerNum, sAscii);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM Serial as String
```

Specific Prototype

<Not Implemented>

Generic Prototype

```
Serial = ic.PropertyString(JSR_ID_PulserHardwareSerNum)
```

11.5 JSR_ID_PulserTriggerEnable

Data Type	Count	Attributes	Object
JSR_Bool	1	Read - Write	Pulser

Contains a JSR_Bool value which controls the on/off state of generating pulses. For this and other Enable/Disable properties developers can cast the boolean into the JSR_EnableEnum type. User text for these can be retrieved via the listID (which is usually JSR_ID_ReferenceEnableNames) or the ID's JSR_ID_ReferenceFalseTrueNames and JSR_ID_ReferenceOffOnNames.

The power-up default value is JSR_DISABLE (pulser off). The Host Application must set this value to JSR_ENABLE for the JSR instrument to generate pulses.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Bool bState;
```

Prototypes

```
JSR_Status JSR_GetBool(nHandle, JSR_ID_PulserTriggerEnable, 1, &bState);
JSR_Status JSR_SetBool(nHandle, JSR_ID_PulserTriggerEnable, 1, bState);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Bool bState;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetBool(nHandle, JSR_ID_PulserTriggerEnable, 1, &bState);
JSR_Status JSRLibFuncs.SetBool(nHandle, JSR_ID_PulserTriggerEnable, 1, bState);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Bool bState;
```

Specific Prototypes

```
JSR_Status ic.GetPulserTriggerEnable(&bState);
JSR_Status ic.SetPulserTriggerEnable(bState);
```

Generic Prototypes

```
JSR_Status ic.GetBool(JSR_ID_PulserTriggerEnable, &bState);
JSR_Status ic.SetBool(JSR_ID_PulserTriggerEnable, bState);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM bState as Bool
```

Specific Prototype

```
bState = ic.PulserTriggerEnable
ic.PulserTriggerEnable = bState
```

Generic Prototype

```
bState = ic.PropertyBool(JSR_ID_PulserTriggerEnable)
ic.PropertyBool(JSR_ID_PulserTriggerEnable) = bState
```

11.6 JSR_ID_PulserIsPulsing

Data Type	Count	Attributes	Object
JSR_Bool	1	Read - Volatile	Pulser

Contains a Boolean value which indicates whether the pulser circuit is generating pulses or not. Display text for this can be requested using JSR_EnableEnum and JSR_ID_ReferenceEnableNames.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Bool bState;
```

Prototypes

```
JSR_Status JSR_GetBool(nHandle, JSR_ID_PulserIsPulsing, 1, &bState);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Bool bState;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetBool(nHandle, JSR_ID_PulserIsPulsing, 1, &bState);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Bool bState;
```

Specific Prototypes

```
JSR_Status ic.GetPulserIsPulsing(&bState);
```

Generic Prototypes

```
JSR_Status ic.GetBool(JSR_ID_PulserIsPulsing, &bState);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM bState as Bool
```

Specific Prototype

```
bState = ic.PulserIsPulsing
```

Generic Prototype

```
bState = ic.PropertyBool(JSR_ID_PulserIsPulsing)
```

11.7 JSR_ID_PulserTriggerSource

Data Type	Count	Attributes	Object
JSR_TriggerSourceEnum	1	Read - Write	Pulser

Controls the selection of the source of the trigger signal to generate pulses. This property is controlled via JSR_TriggerSourceEnum, and display text can be requested using JSR_ID_ReferenceTriggerSourceNames

JSR_TRIGGER_INTERNAL	When set to this mode, the pulser generates pulses based on its internal Pulse Repetition Frequency (JSR_ID_PulserPRF) and the state of JSR_ID_PulserTriggerEnable.
JSR_TRIGGER_EXTERNAL	When set to this mode, the pulser generates one pulse for every trigger signal received by the trigger input connector when JSR_ID_PulserTriggerEnable is JSR_TRUE.
JSR_TRIGGER_SLAVE	When set to this mode, the pulser trigger is slaved to the pulser in the other channel of the DPR500. Only one pulser in one channel can be set to JSR_TRIGGER_SLAVE. An attempt to set two JSR_TRIGGER_SLAVE will return an error. If a DPR500 is configured to have only one pulser, an attempt to set a pulser to JSR_TRIGGER_SLAVE will return an error. JSR_TRIGGER_SLAVE only available for DPR500 with two channels.

Your application can determine at run-time whether a pulser supports two or three modes by sensing the "limitHi" field of the JSR_InfoStruct for this property. The limitHi field will be either JSR_TRIGGER_EXTERNAL, or JSR_TRIGGER_SLAVE.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_TriggerSourceEnum nSource;
```

Prototypes

```
JSR_Status JSR_GetInt32(nHandle, JSR_ID_PulserTriggerSource, 1, &nSource);
JSR_Status JSR_SetInt32(nHandle, JSR_ID_PulserTriggerSource, 1, nSource);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_TriggerSourceEnum nSource;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetInt32(nHandle, JSR_ID_PulserTriggerSource, 1, &nSource);
JSR_Status JSRLibFuncs.SetInt32(nHandle, JSR_ID_PulserTriggerSource, 1, nSource);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_TriggerSourceEnum nSource;
```

Specific Prototypes

```
JSR_Status ic.GetPulserTriggerSource (&bSource);
JSR_Status ic.SetPulserTriggerSource (bSource);
```

Generic Prototypes

```
JSR_Status ic.GetInt32(JSR_ID_PulserTriggerSource, &nSource);
JSR_Status ic.SetInt32(JSR_ID_PulserTriggerSource, nSource);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM Source as JSR_TriggerSourceEnum
```

Specific Prototype

```
Source = ic.PulserTriggerSource
ic.PulserTriggerSource = Source
```

Generic Prototype

```
Source = ic.PropertyInt32(JSR_ID_PulserTriggerSource)
ic.PropertyInt32(JSR_ID_PulserTriggerSource) = Source
```

11.8 JSR_ID_PulserDampResistorList

Data Type	Count	Attributes	Object
JSR_Double array	Variable	Read	Pulser

Contains a list of the damping resistors available in the pulser identified by the handle. The currently selected pulser setting is contained in the property JSR_ID_PulserDampResistorIndex. Neither zero nor infinite resistance is available.

The number of available settings can be found in the elementCount if the InfoStruct via a JSR_GetInfo() function. The element count is constant for a single pulser, but can be different for different pulsers.

This list of resistor values is in the order of decreasing resistance. This is done so the lowest index value of zero provides the minimum amount of damping, (the highest resistance in Ohms). The highest allowed index provides the largest amount of damping (the lowest resistance in Ohms).

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Double fArray[n];
```

Prototypes

```
JSR_Status JSR_GetDouble(nHandle, JSR_ID_PulserDampResistorList, n, fArray);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Double fArray[n];
```

Prototypes

```
JSR_Status JSRLibFuncs.GetDouble(nHandle, JSR_ID_PulserDampResistorList, n, fArray);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Double fArray[n];
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetArrayDouble(JSR_ID_PulserDampResistorList, n, fArray);
```

JSR_Simple_ActiveX.dll

<Not Implemented>

11.9 JSR_ID_PulserDampResistorIndex

Data Type	Count	Attributes	Object
JSR_Int32	1	Read - Write	Pulser

Contains an integer value of the currently selected damping resistor settings, which is specified as an index into the list given by JSR_ID_PulserDampResistorList.

This list of resistor values is in the order of decreasing resistance. This is done so the lowest index value of zero provides the minimum amount of damping, (the highest resistance in Ohms). The highest allowed index provides the largest amount of damping (the lowest resistance in Ohms).

To make this value most robust, it is strongly encouraged to find the 'list' for this ID via the ListID value in an InfoStruct retrieved by a JSR_GetInfo() call on this ID.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Int32 nIndex;
```

Prototypes

```
JSR_Status JSR_GetInt32(nHandle, JSR_ID_PulserDampResistorIndex, 1, &nIndex);
JSR_Status JSR_SetInt32(nHandle, JSR_ID_PulserDampResistorIndex, 1, nIndex);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Int32 nIndex;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetInt32(nHandle, JSR_ID_PulserDampResistorIndex, 1, &nIndex);
JSR_Status JSRLibFuncs.SetInt32(nHandle, JSR_ID_PulserDampResistorIndex, 1, nIndex);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Int32 nIndex;
```

Specific Prototypes

```
JSR_Status ic.GetPulserDampResistorIndex(&bIndex);
JSR_Status ic.SetPulserDampResistorIndex(bIndex);
```

Generic Prototypes

```
JSR_Status ic.GetInt32(JSR_ID_PulserDampResistorIndex, &bSource);
JSR_Status ic.SetInt32(JSR_ID_PulserDampResistorIndex, bSource);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM nIndex as Integer
```

Specific Prototype

```
nIndex = ic.PulserDampResistorIndex
ic.PulserDampResistorIndex = nIndex
```

Generic Prototype

```
nIndex = ic.PropertyInt32(JSR_ID_PulserDampResistorIndex)
ic.PropertyInt32(JSR_ID_PulserDampResistorIndex) = nIndex
```

11.10 JSR_ID_PulserEnergyIndex

Data Type	Count	Attributes	Object
JSR_Int32	1	Read - Write	Pulser

Contains an integer value of the current pulser energy level, which is an index into an Energy list provided by another Property.

This property is somewhat complex, since the property providing the Energy List related to this energy index can change based on the instrument type. For this ID it can be assumed that the values in all Energy Lists are sorted so that an index value of zero provides the lowest energy, and energy increases as the index increases.

To make this ID work properly, the related list must be located via the ListID value in an InfoStruct retrieved by a JSR_GetInfo() call on this ID. These lists may be in many different units of energy, or even in simply a 'High' or 'Low' status.

PRC50 Only The List property for the index is provided via property JSR_ID_PulserEnergyList. The "listID" field of the info struct specifies for this property may specify different list properties, depending on the instrument model. The units used for this should be looked up via the "Units" field of the JSR_InfoStruct for the list ID.

DPR500 and DPR300 The List property for the index is provided via property JSR_ID_ReferenceLowHighNames. There are no units specified in this case.

Like all Properties, a SetInt32() will return an error status if the property's limits are exceeded However, since the average power drawn from the power supply is based on several settings, it is possible to set a value within the allowed limits, but which still exceeds the power limits of the voltage supply. See the description of JSR_ID_PulserPowerLimitStatus for more details on how to detect this case.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Int32 nIndex;
```

Prototypes

```
JSR_Status JSR_GetInt32(nHandle, JSR_ID_PulserEnergyIndex, 1, &nIndex);
JSR_Status JSR_SetInt32(nHandle, JSR_ID_PulserEnergyIndex, 1, nIndex);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Int32 nIndex;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetInt32(nHandle, JSR_ID_PulserEnergyIndex, 1, &nIndex);
JSR_Status JSRLibFuncs.SetInt32(nHandle, JSR_ID_PulserEnergyIndex, 1, nIndex);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Int32 nIndex;
```

Specific Prototypes

```
JSR_Status ic.GetPulserEnergyIndex(&bIndex);
JSR_Status ic.SetPulserEnergyIndex(bIndex);
```

Generic Prototypes

```
JSR_Status ic.GetInt32(JSR_ID_PulserEnergyIndex, &nIndex);
JSR_Status ic.SetInt32(JSR_ID_PulserEnergyIndex, nIndex);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM nIndex as Integer
```

Specific Prototype

```
nIndex = ic.PulserEnergyIndex
ic.PulserEnergyIndex = nIndex
```

Generic Prototype

```
nIndex = ic.PropertyInt32(JSR_ID_PulserEnergyIndex)
ic.PropertyInt32(JSR_ID_PulserEnergyIndex) = nIndex
```

11.11 JSR_ID_PulserPRF

Data Type	Count	Attributes	Object
JSR_Double	1	Read - Write	Pulser

Contains a Double value of the current Pulse Repetition Frequency that is used when in Internal Triggering mode.

This value is used for the power calculations performed by JSR Common API Library in both Internal Triggering and External Triggering mode. These calculations are done when setting or getting values of the power limit Properties: JSR_ID_PulserPowerLimitStatus, JSR_ID_PulserPowerLimitVolts, and JSR_ID_PulserPowerLimitEnergyIndex.

The DPR300 operates on only 16 discrete internal PRF frequencies just like the rotary switch on front panel of the instrument, but the API allows a JSR_Double value for control. When an application commands a PRF to the JSR_Common Library, the Library matches the user's commanded PRF to the closest of the 16 fixed values. The library sets one of those fixed values into the DPR300 and will report that fixed value when the PRF is read from the DPR300.

DPR300 Pulser PRF:
100.0, 200.0, 400.0, 600.0, 800.0, 1000.0, 1250.0, 1500.0,
1750.0, 2000.0, 2500.0, 3000.0, 3500.0, 4000.0, 4500.0, 5000.0

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Double fPRF;
```

Prototypes

```
JSR_Status JSR_GetDouble(nHandle, JSR_ID_PulserPRF, 1, &fPRF);
JSR_Status JSR_SetDouble(nHandle, JSR_ID_PulserPRF, 1, fPRF);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Double fPRF;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetDouble(nHandle, JSR_ID_PulserPRF, 1, &fPRF);
JSR_Status JSRLibFuncs.SetDouble(nHandle, JSR_ID_PulserPRF, 1, fPRF);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Double fPRF;
```

Specific Prototypes

```
JSR_Status ic.GetPulserPRF(&fPRF);
JSR_Status ic.SetPulserPRF(fPRF);
```

Generic Prototypes

```
JSR_Status ic.GetDouble(JSR_ID_PulserPRF, &fPRF);
JSR_Status ic.SetDouble(JSR_ID_PulserPRF, fPRF);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM PRF as Double
```

Specific Prototype

```
PRF = ic.PulserPRF
ic.PulserPRF = PRF
```

Generic Prototype

```
PRF = ic.PropertyDouble(JSR_ID_PulserPRF)
ic.PropertyDouble(JSR_ID_PulserPRF) = PRF
```

11.12 JSR_ID_PulserVolts

Data Type	Count	Attributes	Object
JSR_Double	1	Read - Write	Pulser

Controls the DC high voltage applied to the pulser output circuit. The amplitude of the pulse will be controlled by the DC voltage but will not be identical to it.

The DC high voltage supply may require 1 to 4 seconds to settle to the commanded voltage and may also overshoot the desired voltage during a large change. If your application requires fast response time, you should measure the response time with your specific configuration and your specific voltage changes while developing your code.

A JSR_SetDouble() of JSR_ID_PulserVolts will always return an error status if the limits of voltage are exceeded. However, even if the returned status was JSR_OK, the pulse voltage value could be invalid. Since the average power drawn from the high voltage power supply is a non-linear function of 2 or 3 independent variables, it is possible to set a value to JSR_ID_PulserVolts that will not cause an error status at the time it is set but would still exceed the power limits of the high voltage supply. JSR_ID_PulserPowerLimitStatus may be used to test for that condition.

The DPR300 operates on only 16 discrete internal DC supply voltages to match the rotary switches on the front panel of the DPR300 instrument, but the API allows a JSR_Double value for control. When an application commands a voltage to the JSR_Common Library, the Library matches the user's commanded voltage to the closest of the 16 fixed values. The library sets one of those fixed values into the DPR300 and will report that fixed value when the voltage is read from the DPR300. The DPR300 is available to be purchased with either 475 volts or 900 volts as the maximum value, which will be reported in the limitHi.d field of the information struct for this property.

DPR300 Pulser Volts for 475 V unit:
100.0, 125.0, 150.0, 175.0, 200.0, 225.0, 250.0, 275.0,
300.0, 325.0, 350.0, 375.0, 400.0, 425.0, 450.0, 475.0
DPR300 Pulser Volts for 900 V unit:
100.0, 153.0, 207.0, 260.0, 313.0, 367.0, 420.0, 473.0,
527.0, 580.0, 633.0, 687.0, 740.0, 793.0, 847.0, 900.0

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Double fVolts;
```

Prototypes

```
JSR_Status JSR_GetDouble(nHandle, JSR_ID_PulserVolts, 1, &fVolts);
JSR_Status JSR_SetDouble(nHandle, JSR_ID_PulserVolts, 1, fVolts);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Double fPRF;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetDouble(nHandle, JSR_ID_PulserVolts, 1, &fVolts);
JSR_Status JSRLibFuncs.SetDouble(nHandle, JSR_ID_PulserVolts, 1, fVolts);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Double fVolts;
```

Specific Prototypes

```
JSR_Status ic.GetPulserVolts(&fVolts);
JSR_Status ic.SetPulserVolts(fVolts);
```

Generic Prototypes

```
JSR_Status ic.GetDouble(JSR_ID_PulserVolts, &fVolts);
JSR_Status ic.SetDouble(JSR_ID_PulserVolts, fVolts);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM Volts as Double
```

Specific Prototype

```
Volts = ic.PulserVolts
ic.PulserVolts = Volts
```

Generic Prototype

```
PRF = ic.PropertyDouble(JSR_ID_PulserVolts)
ic.PropertyDouble(JSR_ID_PulserVolts) = PRF
```

11.13 JSR_ID_PulserExtTriggerZList

Data Type	Count	Attributes	Object
JSR_Int32	2	Read	Pulser

This property contains a list of available impedance (Z) settings for the external trigger input. The currently selected Z is contained in the property JSR_ID_PulserExtTriggerZIndex.

The number of available settings can be found in the elementCount if the InfoStruct via a JSR_GetInfo() function. The element count is constant for a single pulser, but can be different for different pulsers. This list Z values are often (but not always) in order, lowest to highest.

Some instruments do not allow input impedance to be changed. When that is the case, the only allowed value will be in the list and JSR_ID_PulserExtTriggerZIndex value will be that one index value.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Int32 nArray[2];
```

Prototypes

```
JSR_Status JSR_GetInt32(nHandle, JSR_ID_PulserExtTriggerZList, 2, nArray);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Int32 nArray[2];
```

Prototypes

```
JSR_Status JSRLibFuncs.GetInt32(nHandle, SR_ID_PulserExtTriggerZList, 2, nArray);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Handle nArray[2];
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetArrayInt32(SR_ID_PulserExtTriggerZList, 2, nArray);
```

JSR_Simple_ActiveX.dll

<Not Implemented>

11.14 JSR_ID_PulserExtTriggerZIndex

Data Type	Count	Attributes	Object
JSR_Int32	1	Read - Write	Pulser

This property contains an integer index of the currently selected impedance(Z) setting for the external trigger input. The value is an index into the list contained by the property JSR_ID_PulserExtTriggerZList.

Some instruments do not allow input impedance to be changed. When that is the case, the only allowed value will be in the list JSR_ID_PulserExtTriggerZList, and this property will have that index value. This property will still have the write attribute set, and so will allow the JSR_SetInt32() command, but only one value will be within the allowed limits. Any attempt to set a different value will cause the return of status code JSR_FAIL_VALUE_TOO_LOW or JSR_FAIL_VALUE_TOO_HIGH.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Int32 nIndex;
```

Prototypes

```
JSR_Status JSR_GetInt32(nHandle, JSR_ID_PulserExtTriggerZIndex, 1, &nIndex);
JSR_Status JSR_SetInt32(nHandle, JSR_ID_PulserExtTriggerZIndex, 1, nIndex);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Int32 nIndex;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetInt32(nHandle, JSR_ID_PulserExtTriggerZIndex, 1, &nIndex);
JSR_Status JSRLibFuncs.SetInt32(nHandle, JSR_ID_PulserExtTriggerZIndex, 1, nIndex);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Int32 nIndex;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetInt32(JSR_ID_PulserExtTriggerZIndex, &nIndex);
JSR_Status ic.SetInt32(JSR_ID_PulserExtTriggerZIndex, nIndex);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM nIndex as Integer
```

Specific Prototype

<Not Implemented>

Generic Prototype

```
nIndex = ic.PropertyInt32(JSR_ID_PulserExtTriggerZIndex)
ic.PropertyInt32(JSR_ID_PulserExtTriggerZIndex) = nIndex
```

11.15 JSR_ID_PulserTriggerEdge

Data Type	Count	Attributes	Object
JSR_Enum	1	Read - Write	Pulser

This property contains an enum value specifying which logic level transition (low to high, or high to low) on the external trigger connection causes the instrument to generate a pulse.

JSR_TriggerEdgeEnum	Action
JSR_TRIGGER_EDGE_RISING	Triggers on rising edge
JSR_TRIGGER_EDGE_FALLING	Triggers on falling edge

Strings that can be used in a GUI that correspond to this property are available from property JSR_ID_ReferenceTriggerEdgeNames.

PRC50 Only This instrument's pulser allow sJSR_TRIGGER_EDGE_RISING and JSR_TRIGGER_EDGE_FALLING. Future JSR products or upgrades to the PRC50 may someday allow more modes.

DPR500 and DPR300 These instruments pulsers report and allow only one setting, JSR_TRIGGER_EDGE_RISING. The property will be writable but will only accept one value.

Applications using this property should determine at run-time how many possible settings are available for each specific pulser. This is done via the JSR_GetInfo() of this ID using the specific pulser handle. The InfoStruct limitHi/limitLo will specify what edge settings are valid.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_TriggerEdgeEnum nState;
```

Prototypes

```
JSR_Status JSR_GetInt32(nHandle, JSR_ID_PulserTriggerEdge, 1, &nState);
JSR_Status JSR_SetInt32(nHandle, JSR_ID_PulserTriggerEdge, 1, nState);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_TriggerEdgeEnum nState;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetInt32(nHandle, JSR_ID_PulserTriggerEdge, 1, &nState);
JSR_Status JSRLibFuncs.SetInt32(nHandle, JSR_ID_PulserTriggerEdge, 1, nState);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_TriggerEdgeEnum nState;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetInt32(JSR_ID_PulserTriggerEdge, &nState);
JSR_Status ic.SetInt32(JSR_ID_PulserTriggerEdge, nState);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM State as JSR_TriggerEdgeEnum
```

Specific Prototype

<Not Implemented>

Generic Prototype

```
State = ic.PropertyInt32(JSR_ID_PulserTriggerEdge)
ic.PropertyInt32(JSR_ID_PulserTriggerEdge) = State
```

11.16 JSR_ID_PulserPowerLimitStatus

Data Type	Count	Attributes	Object
JSR_Status	1	Read - Volatile	Pulser

This property contains a status value of the instruments power draw. It indicates if the power draw is more that available for the voltage supply by returning an error code. The status type returned indicates the success of the GetInt32 call, and the property value returned will indicate the power limit status.

The status will be one of the following values:
JSR_OK or JSR_FAIL_POWER_LIMIT_EXCEEDED.

The high voltage power supply has limits on maximum power output and maximum current output as a function of output voltage. The power that will be drawn from the pulser high voltage supply is a non-linear function of the settings of JSR_ID_PulserVolts, JSR_ID_PulserEnergyIndex, and JSR_ID_PulserPRF.

Since power draw is a function of three properties, setting each property may be within the valid range, but the combination of values may draw more power than is available. To test for this condition, a call to GetInt32() on this property should be done after any of those values are changed.

The power calculations done for this property use the current value of JSR_ID_PulserPRF. When using external triggering, the application should set JSR_ID_PulserPRF to the same value as the expected external PRF. If they are not close, the result of JSR_ID_PulserPowerLimitStatus will be invalid

Exceeding the power limit will not cause any harm to the circuitry. It will result in a pulse output amplitude that 'droops' over a series of pulses. If the power limit is exceeded, the first few pulses will usually have full energy (due to the filtering capacitors in the power supply) but the energy of each later pulse will decrease as a function of time, causing unreliable and non-repeatable pulse waveforms.

The power draw is **not** affected by the selection of the damping resistor.

Some DPR300 models have front panel manual controls. If an operator changed one of those manual controls since the last time the JSR Library set a value to set that control, the two settings could be different. The JSR Common Library therefore reads the values from the DPR300 for all Power Limit functions to ensure the actual settings are used in the calculations

Example Code

```
JSR_Status   powerStatus = JSR_OK;
JSR_Status   returnedStatus = JSR_OK;
returnedStatus = JSR_GetInt32( MyPulserHandle, JSR_ID_PulserPowerLimitStatus, 1, &
  powerStatus);
if ( !JSR_STATUS_OK (returnedStatus) )
       printf("ERROR: JSR_GetInt32 Failed\n");
if ( !JSR_STATUS_OK (powerStatus) )
       printf("ERROR: Power control settings are too high\n");
```

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Status nStatus;
```

Prototypes

```
JSR_Status JSR_GetInt32(nHandle, JSR_ID_PulserPowerLimitStatus, 1, (JSR_Int32*)&nStatus);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Status nStatus;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetInt32(nHandle, JSR_ID_PulserPowerLimitStatus, 1,
(JSR_Int32*)&nStatus);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Status nStatus;
```

Specific Prototypes

```
JSR_Status ic.GetPulserPowerLimitStatus(&nStatus);
```

Generic Prototypes

```
JSR_Status ic.GetInt32(JSR_ID_PulserPowerLimitStatus, (JSR_Int32*)&nStatus);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM Status as JSR_Status
```

Specific Prototype

```
Status = ic.PulserPowerLimitStatus
```

Generic Prototype

```
Status = ic.PropertyInt32(JSR_ID_PulserPowerLimitStatus)
```

11.17 JSR_ID_PulserPowerLimitPRF

Data Type	Count	Attributes	Object
JSR_Double	1	Read - Volatile	Pulser

This property calculates a double value which is the maximum PRF rate that will not drive the voltage supply beyond its normal range.

This property does calculations based on the current power settings (much like JSR_ID_PowerLimitStatus) to calculate the maximum 'in range' PRF that can be set. This maximum is based largely on the current values of JSR_ID_PulserEnergyIndex and JSR_ID_PulserVolts at the time it is called. Any PRF rate below the returned value can be set to JSR_ID_PulserPRF without driving the voltage supply beyond its normal limits.

A JSR_GetDouble() call to this property does not cause any state change on the pulser nor in the pulser object. See JSR_ID_PulserPowerLimitStatus for more information on pulser power limits.

Some DPR300 models have front panel manual controls. If an operator changed one of those manual controls since the last time the JSR Library set a value to set that control, the two settings could be different. The JSR Common Library therefore reads the values from the DPR300 for all Power Limit functions to ensure the actual settings are used in the calculations

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Double fPRF;
```

Prototypes

```
JSR_Status JSR_GetDouble (nHandle, JSR_ID_PulserPowerLimitPRF, 1, &fPRF);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Double fPRF;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetDouble(nHandle, JSR_ID_PulserPowerLimitPRF, 1, &fPRF);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Double fPRF;
```

Specific Prototypes

```
JSR_Status ic.GetPulserPowerLimitPRF(&fPRF);
```

Generic Prototypes

```
JSR_Status ic.GetInt32(JSR_ID_PulserPowerLimitPRF, &fPRF);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM PRF as Double
```

Specific Prototype

```
PRF = ic.PulserPowerLimitPRF
```

Generic Prototype

```
PRF = ic.PropertyInt32(JSR_ID_PulserPowerLimitPRF)
```

11.18 JSR_ID_PulserPowerLimitVolts

Data Type	Count	Attributes	Object
JSR_Double	1	Read - Volatile	Pulser

This property calculates a double value which is the maximum voltage that will not drive the voltage supply beyond its normal range.

This property does calculations based on the current power settings (much like JSR_ID_PulserPowerLimitStatus) to calculate the maximum 'in range' value of JSR_ID_PulserVolts. This maximum is based largely on the current values of JSR_ID_PulserPRF and JSR_ID_PulserEnergyIndex.

If the existing settings of JSR_ID_PulserPRF and JSR_ID_PulserEnergyIndex are set so that even the minimum allowed value of JSR_ID_PulserVolts would cause the power limits to be exceeded, the JSR_GetDouble() call to this value will return an error code (JSR_FAIL_POWER_LIMIT_EXCEEDED).

A JSR_GetDouble() call to this property does not cause any state change on the pulser nor in the pulser object. See JSR_ID_PulserPowerLimitStatus for more information on pulser power limits.

Some DPR300 models have front panel manual controls. If an operator changed one of those manual controls since the last time the JSR Library set a value to set that control, the two settings could be different. The JSR Common Library therefore reads the values from the DPR300 for all Power Limit functions to ensure the actual settings are used in the calculations

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Double fVolts;
```

Prototypes

```
JSR_Status JSR_GetDouble (nHandle, JSR_ID_PulserPowerLimitVolts, 1, &fVolts);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Double fVolts;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetDouble(nHandle, JSR_ID_PulserPowerLimitVolts, 1, &fVolts);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Double fVolts;
```

Specific Prototypes

```
JSR_Status ic.GetPulserPowerLimitVolts(&fPRF);
```

Generic Prototypes

```
JSR_Status ic.GetInt32(JSR_ID_PulserPowerLimitVolts, &fVolts);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM Volts as Double
```

Specific Prototype

```
Volts = ic.PulserPowerLimitPRF
```

Generic Prototype

```
Volts = ic.PropertyInt32(JSR_ID_PulserPowerLimitVolts)
```

11.19 JSR_ID_PulserPowerLimitEnergyIndex

Data Type	Count	Attributes	Object
JSR_Int32	1	Read - Volatile	Pulser

This property calculates an integer value which is the maximum energy index which can be set that will not drive the voltage supply beyond its normal range.

This property does calculations based on the current power settings (much like JSR_ID_PulserPowerLimitStatus) to calculate the maximum 'in range' value of JSR_ID_PulserEnergyIndex. This maximum is based largely on the current values of JSR_ID_PulserPRF and JSR_ID_PulserVolts.

If the existing settings of JSR_ID_PulserPRF and JSR_ID_PulserVolts are set so that even the minimum allowed value of JSR_ID_PulserEnergyIndex would cause the power limits to be exceeded, the GetInt32 call to this value will return an error code (JSR_FAIL_POWER_LIMIT_EXCEEDED).

A JSR_GetInt32( ) call to this property does not cause any state change on the pulser or pulser object. See JSR_ID_PulserEnergyIndex for details on how Energy Index values work. See JSR_ID_PulserPowerLimitStatus for more information on pulser power limits.

Some DPR300 models have front panel manual controls. If an operator changed one of those manual controls since the last time the JSR Library set a value to set that control, the two settings could be different. The JSR Common Library therefore reads the values from the DPR300 for all Power Limit functions to ensure the actual settings are used in the calculations

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Int32 nIndex;
```

Prototypes

```
JSR_Status JSR_GetInt32(nHandle, JSR_ID_PulserPowerLimitEnergyIndex, 1, &nIndex);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Int32 nIndex;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetInt32(nHandle, JSR_ID_PulserPowerLimitEnergyIndex, 1, &nIndex);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Int32 nIndex;
```

Specific Prototypes

```
JSR_Status ic.PulserPowerLimitEnergyIndex(&nIndex);
```

Generic Prototypes

```
JSR_Status ic.GetInt32(JSR_ID_PulserPowerLimitEnergyIndex, &nIndex);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM nIndex as Integer
```

Specific Prototype

```
nIndex = ic.PulserPowerLimitEnergyIndex
```

Generic Prototype

```
nIndex = ic.PropertyInt32(JSR_ID_PulserPowerLimitEnergyIndex)
```

11.20 JSR_ID_PulserEnergyPerPulse

Data Type	Count	Attributes	Object
JSR_Double	1	Read - Volatile	Pulser

This property contains the double value containing the calculated energy used per pulse.

This value is volatile because it will change automatically when the value of JSR_ID_PulserVolts or JSR_ID_PulserEnergyIndex is changed. Applications should re-check this value each time the user updates either JSR_ID_PulserVolts or JSR_ID_PulserEnergyIndex.

The energy per pulse is not a function of JSR_ID_PulserPRF, and does not take into account 'droop' that may happen when voltage supply is driven beyond its range. See JSR_ID_PulserPowerLimitStatus for more information on 'droop'. This value is based on the equation energy per pulse = ½ C V^2.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Double fValue;
```

Prototypes

```
JSR_Status JSR_GetDouble(nHandle, JSR_ID_PulserEnergyPerPulse, 1, &fValue);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Double fValue;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetDouble(nHandle, JSR_ID_PulserEnergyPerPulse, 1, &fValue);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Double fValue;
```

Specific Prototypes

```
JSR_Status ic.PulserEnergyPerPulse (&fValue);
```

Generic Prototypes

```
JSR_Status ic.GetInt32(JSR_ID_PulserEnergyPerPulse, &fValue);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM Value as Double
```

Specific Prototype

```
Value = ic.PulserPowerLimitEnergyIndex
```

Generic Prototype

```
Value = ic.PropertyDouble(JSR_ID_PulserEnergyPerPulse)
```

12 Extended Properties of a Pulser Object

These properties are specific to individual pulser models and will not be present in all pulser models.

An attempt to access a property that does not exist within the selected pulser object will cause a status return of JSR_FAIL_ID_NOT_FOUND.

12.1 JSR_ID_PulserTriggerCount

Data Type	Count	Attributes	Object
JSR_Int32	1	Read	Pulser (PRC50)

NOTE: PRC50 Only

Contains an integer count of the total number of pulses generated by the instrument since the PRC50 board was initialized. The value gets set to zero when the PRC50 board is initialized. The count increments each time the instrument generates a pulse, in both internal and external trigger mode

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Int32 nCount;
```

Prototypes

```
JSR_Status JSR_GetInt32(nHandle, JSR_ID_PulserTriggerCount, 1, &nCount);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Int32 nCount;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetInt32(nHandle, JSR_ID_PulserTriggerCount, 1, &nCount);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Int32 nCount;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetInt32(JSR_ID_PulserTriggerCount, &nCount);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM Count as Integer
```

Specific Prototype

<Not Implemented>

Generic Prototype

```
Count = ic.PropertyInt32(JSR_ID_PulserTriggerCount)
```

12.2 JSR_ID_PulserEnergyList

Data Type	Count	Attributes	Object
JSR_Int32	Variable	Read	Pulser (PRC50)

NOTE: PRC50 Only

Contains a list of the available 'energy per pulse' settings. The currently selected energy level is contained in the property JSR_ID_PulserEnergyIndex.

The number of energy levels in the list can change from pulser to pulser, so it is best to always get the count of elements vial the elementCount entry in an InfoStruct struct (which is updated via the JSR_GetInfo function) The element count is constant for a given pulser. A control index value of zero provides the lowest energy, and energy increases as the index increases. Different instruments may present different units of energy for this property, which will always be specified in the units field of a InfoStruct.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Int32 nArray[n];
```

Prototypes

```
JSR_Status JSR_GetInt32(nHandle, JSR_ID_PulserEnergyList, n, nArray);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Int32 nArray[n];
```

Prototypes

```
JSR_Status JSRLibFuncs.GetInt32(nHandle, JSR_ID_PulserEnergyList, n, nArray);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Int32 nArray[n];
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetArrayInt32(JSR_ID_PulserEnergyList, n, (JSR_Int32*)nArray);
```

JSR_Simple_ActiveX.dll

<Not Implemented>

13 Baseline Properties of a Receiver Object

Every Object has its own Property ID range, consisting of Baseline and Extended Properties. Every Object of a type has every baseline property. This section describes all receiver baseline properties, their values, settings, and information on how they use each property. A receiver object's baseline properties include the "standard properties" of all objects in the JSR Common API Library, as listed above.

All receiver objects also contain the Standard Properties and can access the Reference Properties.

13.1 JSR_ID_ReceiverSignalSelect

Data Type	Count	Attributes	Object
JSR_Enum	1	Read - Write	Receiver

This property reports and allows control of an enum value for the current setting for receiver amplification mode, which corresponds to JSR_SignalSelectEnum.

The available modes for JSR_ReceiverSignalSelect are:

JSR_SIGNAL_SELECT_TR_ECHO	"Transmit/Receive" or "T/R" (also known as "Echo" mode),
JSR_SIGNAL_SELECT_THROUGH	"Through mode", (aka. "Pitch-Catch" mode)
JSR_SIGNAL_SELECT_BOTH	"Both" mode. (PRC50 only)

PRC50

In"T/R" mode, the pulser and receiver circuits in the PRC50 share the "T/R" connector on the rear of the PRC50 card. The PRC50 sends out the generated excitation pulse to the transducer and receives the echo from the same transducer.

In 'Through' mode, a second transducer is connected to the 'Through' connector. The "T/R" connector on the PRC50 sends out the excitation pulse to the first transducer, and the 'Through' connector receives the incoming signal into the PRC50 receiver from the second transducer.

In "Both" mode, the return signal is picked up by both the "T/R" and "Through" transducers, and their signals are combined into a single receiver signal within the PRC50. The gain of each channel can be controlled separately via properties JSR_ID_ReceiverTREchoGainDB and JSR_ID_ReceiverThroughGainDB.

DPR300 and DPR500

There are some remote pulsers used with the DPR500 that allow only one of these modes, while others allow more than one.

This property will always have the read and write attributes. If there is only one valid value, the InfoStruct limitLo.i and limitHi.i will both be set to that one allowed value, allowing a write of that value only.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_SignalSelectEnum nSelect;
```

Prototypes

```
JSR_Status JSR_GetInt32(nHandle, JSR_ID_ReceiverSignalSelect, 1, (JSR_Int32*)&nSelect);
JSR_Status JSR_SetInt32(nHandle, JSR_ID_ReceiverSignalSelect, 1, (JSR_Int32)nSelect);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_SignalSelectEnum nSelect;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetInt32(nHandle, JSR_ID_ReceiverSignalSelect, 1, (JSR_Int32*)&nSelect);
JSR_Status JSRLibFuncs.SetInt32(nHandle, JSR_ID_ReceiverSignalSelect, 1, (JSR_Int32)nSelect);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_SignalSelectEnum nSelect;
```

Specific Prototypes

```
JSR_Status ic.GetPulserTriggerSource (&nSelect);
JSR_Status ic.SetPulserTriggerSource (nSelect);
```

Generic Prototypes

```
JSR_Status ic.GetInt32(JSR_ID_ReceiverSignalSelect, (JSR_Int32*)&nSelect);
JSR_Status ic.SetInt32(JSR_ID_ReceiverSignalSelect, (JSR_Int32)nSelect);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM nSelect as JSR_SignalSelectEnum
```

Specific Prototype

```
nSelect = ic.PulserTriggerSource
ic.PulserTriggerSource = nSelect
```

Generic Prototype

```
nSelect = ic.PropertyInt32(JSR_ID_PulserTriggerSource)
ic.PropertyInt32(JSR_ID_PulserTriggerSource) = nSelect
```

13.2 JSR_ID_ReceiverModelName

Data Type	Count	Attributes	Object
JSR_String or JSR_Ascii	1	Read	Receiver

Contains a String value that identifies the receiver model.

Model	displayText	Example	Simulated string
PRC50	“Model”	“PRC50”	“Sim Rec Model”
DPR500	“Model”	“RL-01”	“Sim RL-01”

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_String sString;
JSR_Ascii sAscii;
```

Prototypes

```
JSR_Status JSR_GetString(nHandle, JSR_ID_ReceiverModelName, 1, sString);
JSR_Status JSR_GetAscii(nHandle, JSR_ID_ReceiverModelName, 1, sAscii);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_String sString;
JSR_Ascii sAscii;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetString(nHandle, JSR_ID_ReceiverModelName, 1, sString);
JSR_Status JSRLibFuncs.GetAscii(nHandle, JSR_ID_ReceiverModelName, 1, sAscii);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_String sString;
JSR_Ascii sAscii;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetString(JSR_ID_ReceiverModelName, sString);
JSR_Status ic.GetString(JSR_ID_ReceiverModelName, sAscii);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM Name as String
```

Specific Prototype

<Not Implemented>

Generic Prototype

```
Name = ic.PropertyString(JSR_ID_ReceiverModelName)
```

13.3 JSR_ID_ReceiverSerNum

Data Type	Count	Attributes	Object
JSR_String or JSR_Ascii	1	Read	Receiver

Reports a string value that identifies the receiver serial number or P/R Board serial number.

The displayText field of the InfoStruct for this property is different for different instruments:

Model	displayText	Example	Simulated string
PRC50	“P/R Board SN”	“PR0001”	“Sim P/R Board SN”
DPR500	“Hardware Serial Number”	“9735B”	“Sim Ch A Rec SN”

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_String sString;
JSR_Ascii sAscii;
```

Prototypes

```
JSR_Status JSR_GetString(nHandle, JSR_ID_ReceiverSerNum, 1, sString);
JSR_Status JSR_GetAscii(nHandle, JSR_ID_ReceiverSerNum, 1, sAscii);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_String sString;
JSR_Ascii sAscii;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetString(nHandle, JSR_ID_ReceiverSerNum, 1, sString);
JSR_Status JSRLibFuncs.GetAscii(nHandle, JSR_ID_ReceiverSerNum, 1, sAscii);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_String sString;
JSR_Ascii sAscii;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetString(JSR_ID_ReceiverSerNum, sString);
JSR_Status ic.GetString(JSR_ID_ReceiverSerNum, sAscii);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM Serial as String
```

Specific Prototype

<Not Implemented>

Generic Prototype

```
Serial = ic.PropertyString(JSR_ID_ReceiverSerNum)
```

13.4 JSR_ID_ReceiverHardwareRev

Data Type	Count	Attributes	Object
JSR_String or JSR_Ascii	1	Read	Instrument

Contains a string value that identifies the hardware revision.

Examples: “A”, “B”, “C”

Model	displayText	Example	Simulated string
PRC50	“P/R Board Rev”	“A”	“Sim Rec Rev”
DPR500	“Revision”	“C”	“Sim Ch A Rec Rev”

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_String sString;
JSR_Ascii sAscii;
```

Prototypes

```
JSR_Status JSR_GetString(nHandle, JSR_ID_ReceiverHardwareSerNum, 1, sString);
JSR_Status JSR_GetAscii(nHandle, JSR_ID_ReceiverHardwareSerNum, 1, sAscii);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_String sString;
JSR_Ascii sAscii;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetString(nHandle, JSR_ID_ReceiverHardwareSerNum, 1, sString);
JSR_Status JSRLibFuncs.GetAscii(nHandle, JSR_ID_ReceiverHardwareSerNum, 1, sAscii);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_String sString;
JSR_Ascii sAscii;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetString(JSR_ID_ReceiverHardwareSerNum, sString);
JSR_Status ic.GetString(JSR_ID_ReceiverHardwareSerNum, sAscii);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM Serial as String
```

Specific Prototype

<Not Implemented>

Generic Prototype

```
Serial = ic.PropertyString(JSR_ID_ReceiverHardwareSerNum)
```

13.5 JSR_ID_ReceiverBandwidth

Data Type	Count	Attributes	Object
JSR_Int32	1	Read	Receiver

Contains an integer value for the bandwidth of the channel in MHz.

For example, for the PRC50, a 50Mhz bandwidth instrument, this will be 50. DPR300 and DPR500 instruments can be configured with different receiver boards, so this value should be read from the receiver object at run time.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Int32 nBandwidth;
```

Prototypes

```
JSR_Status JSR_GetInt32(nHandle, JSR_ID_ReceiverBandwidth, 1, &nBandwidth);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Int32 nBandWidth;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetInt32(nHandle, JSR_ID_ReceiverBandwidth, 1, &nBandwidth);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Int32 nBandWidth;
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetInt32(JSR_ID_ReceiverBandwidth, &nBandwidth);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM Bandwidth as Integer
```

Specific Prototype

<Not Implemented>

Generic Prototype

```
Bandwidth = ic.PropertyInt32(JSR_ID_ReceiverBandwidth)
```

13.6 JSR_ID_ReceiverGainDB

Data Type	Count	Attributes	Object
JSR_Double	1	Read - Write	Receiver

This property contains the current gain setting of the Receiver amplifier. Changing this property adjusts the output signal amplitude from the Receiver output signal SMA connector.

NOTE: PRC50 Only

The PRC50 contains 2 amplifiers, a “Transmit / Receive”(aka “T/R” or “Echo”) amplifier, and a “Through” amplifier. These are connected to serrate SMA connectors on the back of the PRC50 board. In normal PRC50 operation, setting JSR_ID_ReceiverGainDB sets the amplification for both of these amplifiers.

This baseline property is available alongside the PRC50 specific gain properties. See the descriptions of JSR_ID_ReceiverTREchoGainDB and JSR_ID_ReceiverThroughGainDB, and the PRC Receiver Block Diagram for more details.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Double fGain;
```

Prototypes

```
JSR_Status JSR_GetDouble(nHandle, JSR_ID_ReceiverGainDB, 1, &fGain);
JSR_Status JSR_SetDouble(nHandle, JSR_ID_ReceiverGainDB, 1, fGain);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Double fPRF;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetDouble(nHandle, JSR_ID_ReceiverGainDB, 1, &fGain);
JSR_Status JSRLibFuncs.SetDouble(nHandle, JSR_ID_ReceiverGainDB, 1, fGain);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Double fPRF;
```

Specific Prototypes

```
JSR_Status ic.GetReceiverGainDB (&fGain);
JSR_Status ic.SetReceiverGainDB (fGain);
```

Generic Prototypes

```
JSR_Status ic.GetDouble(JSR_ID_ReceiverGainDB, &fGain);
JSR_Status ic.SetDouble(JSR_ID_ReceiverGainDB, fGain);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM Gain as Double
```

Specific Prototype

```
Gain = ic.ReceiverGainDB
ic.ReceiverGainDB = Gain
```

Generic Prototype

```
Gain = ic.PropertyDouble(JSR_ID_ReceiverGainDB)
ic.PropertyDouble(JSR_ID_ReceiverGainDB) = Gain
```

13.7 JSR_ID_ReceiverLPFilterList

Data Type	Count	Attributes	Object
JSR_Double	Variable	Read	Receiver

Contains a list of the low pass (LP) filters available on the receiver. The currently selected low pass filter is determined by JSR_ID_ReceiverLPFilterIndex. The units and element count are identified in the JSR_InfoStruct

This list can vary between instruments or channels within an instrument, but remains once a receiver is opened.

The count of available elements is in the elementCount field after a JSR_GetInfo(). The element count is constant for a given JSR receiver version.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Double fArray[n];
```

Prototypes

```
JSR_Status JSR_GetDouble(nHandle, JSR_ID_ReceiverLPFilterList, n, fArray);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Double fArray[n];
```

Prototypes

```
JSR_Status JSRLibFuncs.GetDouble(nHandle, JSR_ID_ReceiverLPFilterList, n, fArray);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Double fArray[n];
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetArrayDouble(JSR_ID_ReceiverLPFilterList, n, fArray);
```

JSR_Simple_ActiveX.dll

<Not Implemented>

13.8 JSR_ID_ReceiverLPFilterIndex

Data Type	Count	Attributes	Object
JSR_Int32	1	Read - Write	Receiver

Contains the currently selected low pass (LP) filter for the receiver. This value is an index into the list of available filters provided by JSR_ID_ReceiverLPFilterList.

The minimum index value is 0 and the maximum value is defined in the property's InfoStruct.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Int32 nIndex;
```

Prototypes

```
JSR_Status JSR_GetInt32(nHandle, JSR_ID_RecieverLPFilterIndex, 1, &nIndex);
JSR_Status JSR_SetInt32(nHandle, JSR_ID_RecieverLPFilterIndex, 1, nIndex);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Int32 nIndex;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetInt32(nHandle, JSR_ID_RecieverLPFilterIndex, 1, &nIndex);
JSR_Status JSRLibFuncs.SetInt32(nHandle, JSR_ID_RecieverLPFilterIndex, 1, nIndex);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Int32 nIndex;
```

Specific Prototypes

```
JSR_Status ic.GetRecieverLPFilterIndex (&bIndex);
JSR_Status ic.SetRecieverLPFilterIndex (bIndex);
```

Generic Prototypes

```
JSR_Status ic.GetInt32(JSR_ID_RecieverLPFilterIndex, &bSource);
JSR_Status ic.SetInt32(JSR_ID_RecieverLPFilterIndex, bSource);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM nIndex as Integer
```

Specific Prototype

```
nIndex = ic.RecieverLPFilterIndex
ic.RecieverLPFilterIndex = nIndex
```

Generic Prototype

```
nIndex = ic.PropertyInt32(JSR_ID_RecieverLPFilterIndex)
ic.PropertyInt32(JSR_ID_RecieverLPFilterIndex) = nIndex
```

13.9 JSR_ID_ReceiverHPFilterList

Data Type	Count	Attributes	Object
JSR_Double	Variable	Read	Receiver

Contains the list of the high pass (HP) filters available on the receiver section of the JSR channel. The units and element count are identified in the J JSR_InfoStruct.

This list can vary between instruments or channels within an instrument, but remains constant once a receiver is opened.

The count of available elements is in the elementCount field after a JSR_GetInfo(). The element count is constant for a given JSR receiver version.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Double fArray[n];
```

Prototypes

```
JSR_Status JSR_GetDouble(nHandle, JSR_ID_ReceiverHPFilterList, n, fArray);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Double fArray[n];
```

Prototypes

```
JSR_Status JSRLibFuncs.GetDouble(nHandle, JSR_ID_ReceiverHPFilterList, n, fArray);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Double fArray[n];
```

Specific Prototypes

<Not Implemented>

Generic Prototypes

```
JSR_Status ic.GetArrayDouble(JSR_ID_JSR_ID_ReceiverHPFilterList, n, fArray);
```

JSR_Simple_ActiveX.dll

<Not Implemented>

13.10 JSR_ID_ReceiverHPFilterIndex

Data Type	Count	Attributes	Object
JSR_Int32	1	Read-Write	Receiver

Contains the currently selected high pass (HP) filter for the receiver. This value is an index into the list of available filters provided by JSR_ID_ReceiverHPFilterList.

The minimum index value is 0 and the maximum value is defined in the property's InfoStruct.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Int32 nIndex;
```

Prototypes

```
JSR_Status JSR_GetInt32(nHandle, JSR_ID_RecieverHPFilterIndex, 1, &nIndex);
JSR_Status JSR_SetInt32(nHandle, JSR_ID_RecieverHPFilterIndex, 1, nIndex);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Int32 nIndex;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetInt32(nHandle, JSR_ID_RecieverHPFilterIndex, 1, &nIndex);
JSR_Status JSRLibFuncs.SetInt32(nHandle, JSR_ID_RecieverHPFilterIndex, 1, nIndex);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Int32 nIndex;
```

Specific Prototypes

```
JSR_Status ic.GetRecieverLPFilterIndex (&bIndex);
JSR_Status ic.SetRecieverLPFilterIndex (bIndex);
```

Generic Prototypes

```
JSR_Status ic.GetInt32(JSR_ID_RecieverHPFilterIndex, &bSource);
JSR_Status ic.SetInt32(JSR_ID_RecieverHPFilterIndex, bSource);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM nIndex as Integer
```

Specific Prototype

```
nIndex = ic.RecieverHPFilterIndex
ic.RecieverHPFilterIndex = nIndex
```

Generic Prototype

```
nIndex = ic.PropertyInt32(JSR_ID_RecieverHPFilterIndex)
ic.PropertyInt32(JSR_ID_RecieverHPFilterIndex) = nIndex
```

14 Extended Properties of a Receiver Object

These properties are specific to individual receiver models and will not be present in all receiver models.

An attempt to access a property that does not exist within the selected receiver object will cause a status return of JSR_FAIL_ID_NOT_FOUND.

14.1 JSR_ID_ReceiverTREchoGainDB

Data Type	Count	Attributes	Object
JSR_Double	1	Read - Write	Receiver (PRC50)

NOTE: PRC50 Only

This property contains the current gain settings for receiver amplification circuit connected to the “T/R” connector on the PRC50, as shown in the PRC50 Receiver Block Diagram. This amplification path is referred to as the “Transmit/Receive”, “T/R” or “Echo” amplification.

This property, along with the JSR_ID_ReceiverThroughGainDB, allow separate gain control of the two amplifier paths. If it is desirable to have your application control the gain of both receiver amplifier circuits simultaneously, you should use the property JSR_ID_ReceiverGainDB.

Use of both the general and specific receiver controls is discouraged, and in PRC50 JSR_ID_ReceiverGainDB has an extra attribute(JSR_ATTRIB_SUPERSEDED) to indicate the extended properties are available.

As with all properties, the minimum and maximum values are specified in the JSR_InfoStruct limitLo and limitHi fields. This property has a negative minimum value.

The T/R connector and some circuitry is shared by the pulse generation circuit of the pulser, as shown in the PRC50 Receiver Block Diagram Because of that the T/R receiver gain also is dependent on the current damping resistor selection. The choice of damping resistance will affect the actual gain of the “T/R” receiver amplifiers.

The “T\R” receiver amplification is calibrated with the Pulser damping resistance of 50 Ohms selected, and the software does NOT compensate for any other damping resistance. Therefore the actual gain will also be a function of the selected damping resistance. See the PRC Receiver Block Diagram for more information.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Double fGain;
```

Prototypes

```
JSR_Status JSR_GetDouble(nHandle, JSR_ID_ReceiverTREchoGainDB, 1, &fGain);
JSR_Status JSR_SetDouble(nHandle, JSR_ID_ReceiverTREchoGainDB, 1, fGain);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Double fPRF;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetDouble(nHandle, JSR_ID_ReceiverTREchoGainDB, 1, &fGain);
JSR_Status JSRLibFuncs.SetDouble(nHandle, JSR_ID_ReceiverTREchoGainDB, 1, fGain);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Double fPRF;
```

Specific Prototypes

```
JSR_Status ic.GetReceiverTREchoGainDB (&fGain);
JSR_Status ic.SetReceiverTREchoGainDB (fGain);
```

Generic Prototypes

```
JSR_Status ic.GetDouble(JSR_ID_ReceiverTREchoGainDB, &fGain);
JSR_Status ic.SetDouble(JSR_ID_ReceiverTREchoGainDB, fGain);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM Gain as Double
```

Specific Prototype

```
Gain = ic.ReceiverTREchoGainDB
ic.ReceiverTREchoGainDB = Gain
```

Generic Prototype

```
Gain = ic.PropertyDouble(JSR_ID_ReceiverTREchoGainDB)
ic.PropertyDouble(JSR_ID_ReceiverTREchoGainDB) = Gain
```

14.2 JSR_ID_ReceiverThroughGainDB

Data Type	Count	Attributes	Object
JSR_Double	1	Read - Write	Receiver (PRC50)

NOTE: PRC50 Only

This property contains the current gain settings for receiver amplification circuit connected to the “Through” connector on the PRC50, as shown in the PRC50 Receiver Block Diagram.

As with all properties, the minimum and maximum values are specified in the JSR_InfoStruct limitLo and limitHi fields. This property has a negative minimum value.

This amplification path is linked to the “T\R” amplification path, as shown in the PRC50 Receiver Block Diagram.

This property, along with the JSR_ID_ReceiverTREchoGainDB, provide separate controls that can replace JSR_ID_ReceiverGainDB. If it is desirable to have your application control the gain of both receiver amplifier circuits simultaneously, you should use the property JSR_ID_ReceiverGainDB. See the PRC Receiver Block Diagram for more information.

JSR_Common.lib

Declarations

```
JSR_Handle nHandle;
JSR_Double fGain;
```

Prototypes

```
JSR_Status JSR_GetDouble(nHandle, JSR_ID_ReceiverThroughGainDB, 1, &fGain);
JSR_Status JSR_SetDouble(nHandle, JSR_ID_ReceiverThroughGainDB, 1, fGain);
```

JSR_Common.dll

Declarations

```
JSR_FunctionPtrs JSRLibFuncs;
JSR_Handle nHandle;
JSR_Double fPRF;
```

Prototypes

```
JSR_Status JSRLibFuncs.GetDouble(nHandle, JSR_ID_ReceiverThroughGainDB, 1, &fGain);
JSR_Status JSRLibFuncs.SetDouble(nHandle, JSR_ID_ReceiverThroughGainDB, 1, fGain);
```

JSR_Simple.lib

Declarations

```
JSR_Simple ic;
JSR_Double fPRF;
```

Specific Prototypes

```
JSR_Status ic.GetReceiverThroughGainDB (&fGain);
JSR_Status ic.SetReceiverThroughGainDB (fGain);
```

Generic Prototypes

```
JSR_Status ic.GetDouble(JSR_ID_ReceiverThroughGainDB, &fGain);
JSR_Status ic.SetDouble(JSR_ID_ReceiverThroughGainDB, fGain);
```

JSR_Simple_ActiveX.dll

Declarations

```
DIM ic as JSR_Simple
DIM Gain as Double
```

Specific Prototype

```
Gain = ic.ReceiverThroughGainDB
ic.ReceiverThroughGainDB = Gain
```

Generic Prototype

```
Gain = ic.PropertyDouble(JSR_ID_ReceiverThroughGainDB)
ic.PropertyDouble(JSR_ID_ReceiverThroughGainDB) = Gain
```

15 Use of Properties with the volatile attribute

Some properties in the JSR Library have an attribute of ‘Volatile’. This indicates that the property can change without direct application input to this property. This may be due to an external event, or that this property is a function of the settings of other properties.

For the most timely information of its state, volatile properties should be updated each time a property changes, as well as checked periodically.

Those controllable properties that affect the results of the several Power Limit properties will have the JSR_ATTRIB_AFFECTS_POWER_LIMIT attribute bit set.

Those properties which have values that can be changed on the front panel of the instrument (currently DPR300’s) will have the JSR_ATTRIB_MANUAL attribute bit set.

16 Auto-Population of Object data via Properties

Application code may have all property identifiers hard coded by the identifier name (e.g. JSR_ID_PulserPRF) to access specific properties one at a time. However, this does not allow for easy upgrades of the API by simply replacing the underlying DLL. For more robust, and easily upgraded application, we encourage dynamically generating an interface based on built-in properties.

Applications can auto-populate the list of all properties by doing the following

1) Get the number of properties of the selected object by calling JSR_GetInfo() using the object handle and the ID of JSR_ID_AvailablePropertyIDs. The elementCount value of the JSR_InfoStruct, will indicate the number of properties available

2) Allocate an array of JSR_PropID items based on the count of elements from number 1. Then to get the actual list of all object properties, call JSR_GetInt32() using the object handle and JSR_ID_AvailablePropertyIDs.

3) If the JSR_GetInt32() function was successful, loop through the array of JSR_PropID items, and use JSR_GetInfo() to collect display information such as type, size, precision, and attributes.

Beware of automatically opening and expanding new objects are they are encountered, it can easily lead to infinite loop problems. Your application should **never** automatically expand and 'walk' Object ID's returned by JSR_ID_ParentHandle. By definition, the parent handle of the Library is itself, which will result in an endless loop.

Reference ID's are available to ALL objects, but are not listed by JSR_ID_AvailablePropertyID's. The section "Property Overview", above, describes groupings of properties.

17 Using Enum's and Name Property ID's

For quicker development of GUI applications and to allow the greatest amount of language support built into the library, the JSR Library has a system for using Properties and Enum's together to quickly return a text string representing an Enumeration.

Example:

An application developer wishes the application to display the Object type (Instrument/Pulser-Receiver/Etc) at the top of a window contain information on that object. To do this, the application code:

1) Calls JSR_GetInfo() with the second argument JSR_ID_ObjectType to get the InfoStruct

2) The listID value of the returned JSR_InfoStruct will contain the non-zero Property ID which in this case will be JSR_ID_ReferenceObjectTypeNames. The application then calls JSR_GetInfo(JSR_ID_ReferenceObjectTypeNames) to determine the element count of the JSR_ID_ReferenceObjectTypeNames property.

3) The application then calls JSR_GetString(JSR_ID_ReferenceObjectTypeNames) to get an array of strings of the names.

4) Calls JSR_GetInt32(JSR_ID_ObjectType) with to get the value of the object type enum.

5) The application uses the object type enum returned above as an index into the array of strings.

18 Forward Compatibility

Developers of the JSR Common API have worked hard to generate a library where the vast majority of controls and information are available by runtime requests, in order to make our library forward compatible.

Enumerations, property ranges and options, and text strings may all be expanded in future releases of the JSR Common API. By creating a robust program that does strong error checking, and requests all information via elementCount, limitLo, limitHi, etc. an application can be developed with is upgradeable by simply dragging and dropping a new DLL into place.

19 Error Code Ranges

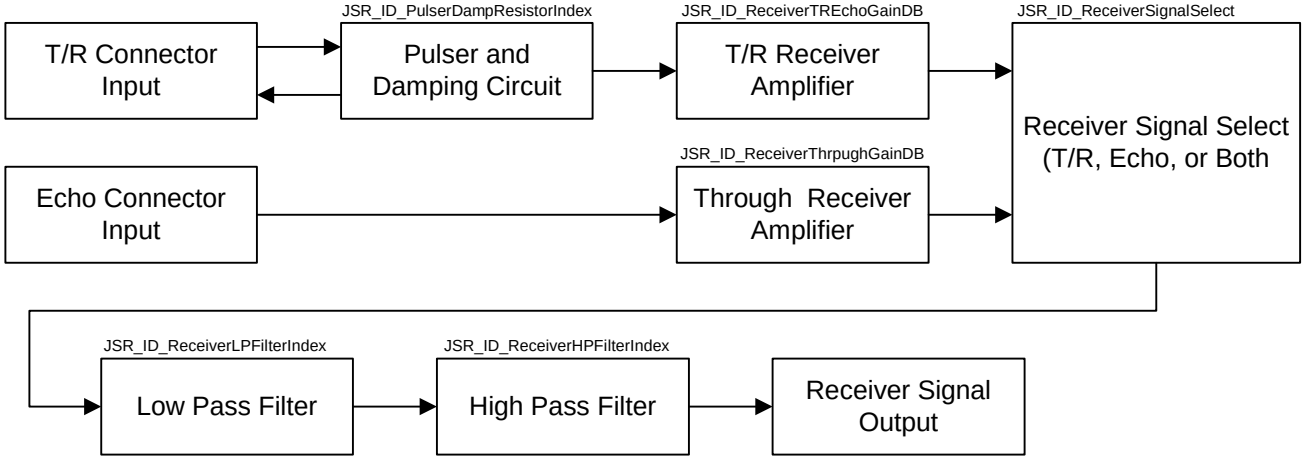
The JSR Common API error code range is designed to make it easy to identify fatal (“FAIL”) or non-fatal (“WARN”) status return values. There is a gap of numbers which can be used by 3rd party applications.

ID Name	Value(s)	How it’s used
JSR_OK	0	This is always used to indicate ‘Function completed OK’
none specified	1 –1023	This range is not used by, it is reserved for 3rd party developers use
JSR_WARN_XXX	1024-2047	This range is warnings, non-fatal error. Function completed with problems
JSR_ERR_XXX	2048 ->	This range is ‘fatal’ errors. The function did not complete successfully

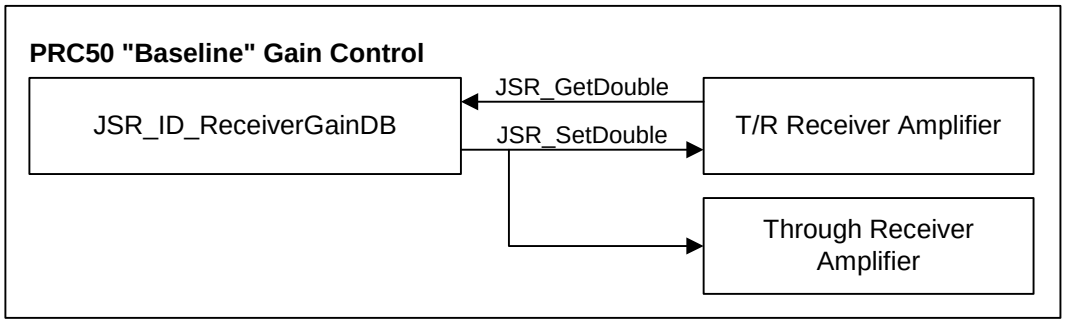
All of these error codes can be used as input for JSR_GetError(), which will fill the passed in a text string with a proper error string. For JSR_Status values from 1 – 1023, that error string will say “Application Error #X”, where X is the integer value of the JSR_Status.

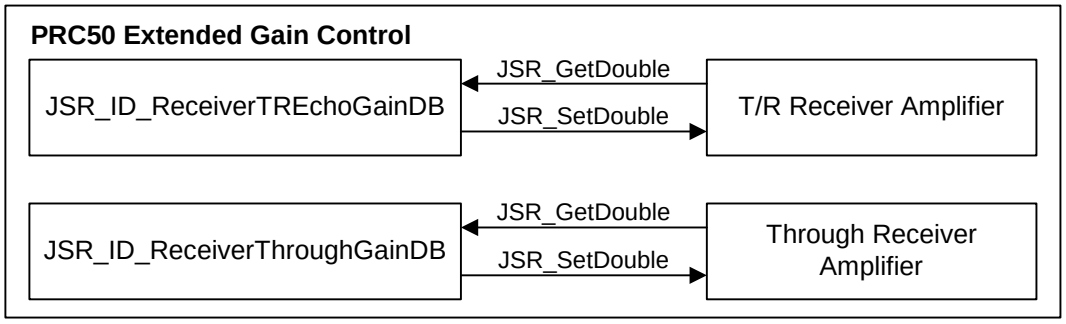
20 PRC50 Receiver Block Diagram

Overview of PRC50 Receiver



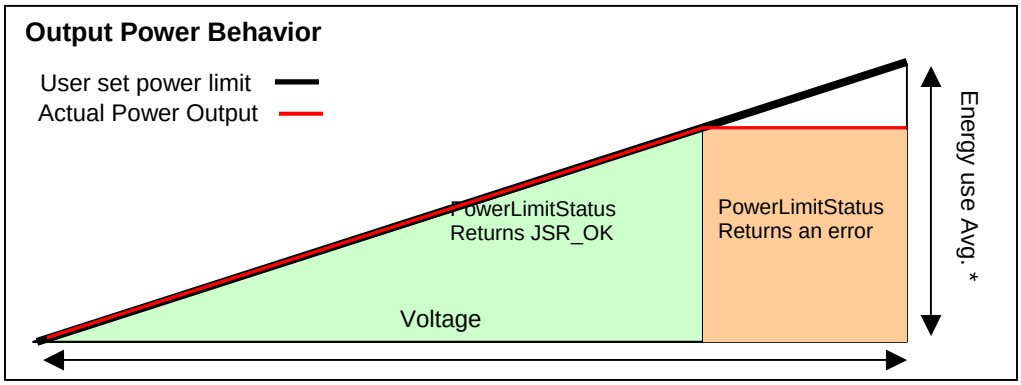
Overview of PRC50 Gain Modes





21 PRC50 Pulser Power Limit Details

The Power Limit system on the PRC50 is specifically designed to allow developers and users to overdrive the power supply in some cases. To accommodate this, the power limit system is more robust and in depth than simply' set a value and get an error if it wrong. This section details how the Power Limit system works.



* Energy Use Avg. is a function of the PRF and EnergyIndex settings

The Output Power Behavior graph is a simplification of the actual power output, created as an example.

Another way to view the same information is via a table. The below table is an example graph of how to understand the value in JSR_ID_PulserPowerLimitPRF. These values are given as an example as a guideline for understanding the "Power Limit" properties. The values from the table below should not be put into application code for actual use.

An application should perform a "JSR_Get…" of the appropriate Power Limit property and should not use the graph above or the numbers from the table below.

Example Table:

		EnergyIndex						
		330	660	990	1320	1650	1980	2310
Volts	min(i)	PowerLimitPRF in Hz						
100	0.0400	5000	5000	5000	5000	5000	5000	5000
250	0.0150	5000	5000	5000	5000	5000	5000	5000
300	0.0100	5000	5000	5000	5000	5000	5000	5000
350	0.0075	5000	5000	5000	5000	5000	5000	5000
400	0.0050	5000	5000	5000	5000	5000	5000	5000
450	0.0045	5000	5000	5000	5000	5000	5000	4329
475	0.0043	5000	5000	5000	5000	5000	4519	3873

22 DPR500 Remote Pulser Unit Notes

The DPR500 has remote pulser units which are interchangeable. These remote pulser units should not be swapped with one another while the SDK is open and running.

For software to detect and identify the pulser units properly:

Close the entire JSR_Common Library

Turn power off to the DPR500

Disconnect the Remote Pulser.

Connect the new Remote Pulser.

Turn on power to the DPR500

Open the JSR Library,

Open the Instrument Object in software

Open the Channel Object in software

Open the Pulser Object in software

DPR remote pulsers will not be correctly detected if:

1) A remote pulser gets connected after power-up or after the Library is Opened
2) A remote pulser gets swapped for a different one while the Library is open
3) A remote pulser gets removed without a restart. When the pulser is removed without restart, it may cause errors for other items on the channel or instrument.

23 Use of List ID in Properties

The JSR_InfoStruct member List ID is used to allow properties to depend on data in other properties. This allows complex behavior based on dependent properties. This system also allows the application developer's package to be updated by a simple replacement of the JSR Common DLL, without recompiling the developer's application.

List ID's have two ways of being used.

1) ID X is an index into its list ID (List ID as Lookup)
2) ID X is a list of values, ordered to match the items in its list ID (List ID as Order)

23.1 List ID as Lookup of a value

When using list ID as a lookup system, the property X ID contains an integer index, and the List ID contains a list of values. The integer index indicates which of those values to select. This type of ID behavior is specified by having a non-zero List ID, and the attribute JSR_ATTRIB_INDEX.

What is needed for List ID as Lookup

Property X has a List ID that is not zero

Property X has a single element (elementCount of 1)

Property X has the attribute JSR_ATTRIB_INDEX

A simple example of this mechanism is the connection between properties JSR_ID_PulserDampResistorIndex and JSR_ID_PulserDampResistorList. JSR_ID_PulserDampResistorList is a read-only list of values of the resistors available in the selected pulser and JSR_ID_PulserDampResistorIndex is the read-write value the application controls to select one specified resistor value.

23.2 List ID as Lookup of an enumerated string

A simple example of this mechanism is the connection between properties JSR_ID_ReceiverSignalSelect and JSR_ID_ReferenceSignalSelectNames. JSR_ID_ReferenceSignalSelectNames is a read-only list of strings that describe each optional setting and JSR_ID_ReceiverSignalSelect is the read-write value the application controls to select one specified signal selection mode.

There are a number of controllable (read-write) properties that have only two possible settings, represented by the integer values zero and one. Such properties may have a non-zero listID that identifies a specific reference but any other reference string with two cases can be used if more intuitive to the user.