

# Digilent Adept General Sensor and User Interface Device (DGIO) Programmer's Reference Manual



Revision: August 16, 2010

1300 NE Henley Court, Suite 3  
Pullman, WA 99163  
(509) 334 6306 Voice | (509) 334 6300 Fax

## Introduction

This document describes the Digilent Adept General Sensor and User Interface Device (DGIO) subsystem for version 2 of the Digilent Adept software system. This document describes the capabilities of the DGIO subsystem and the API functions used to access it.

The DGIO subsystem provides a generic interface to various kinds of sensor input devices and user input/output devices. Sensor devices accessible through this subsystem include such things as accelerometers, pressure transducers, gyroscopes, ultra-sonic range finders, limit switches and many others. The user interface devices include such things as buttons and switches, discrete LEDs, keyboards, keypads, joysticks, rotary encoders and so on.

## DGIO Port Properties

The port property bits are used to indicate the specific capabilities or features of the DGIO interface implemented by a port.

The following properties are defined for DGIO ports: These values are defined in the header file `dgio.h`.

<code>dprpGioDiscreteIn</code>	The port/channel provides discrete (bitwise) inputs.
<code>dprpGioDiscreteOut</code>	The port/channel provides discrete (bitwise) outputs.
<code>dprpGioRangedIn</code>	The port/channel provides ranged inputs.
<code>dprpGioRangedOut</code>	The port/channel provides ranged outputs.
<code>dprpGioEncodedIn</code>	The port/channel provides encoded inputs. (Reserved for future use)
<code>dprpGioEncodedOut</code>	The port/channel provides encoded outputs. (Reserved for future use)
<code>dprpGioAbsolute</code>	The port/channel supports absolute ranged input/output. This bit will only be set if <code>dprpGioRangedIn</code> or <code>dprpGioRangedOut</code> is set.
<code>dprpGioRelative</code>	The port/channel supports relative ranged input/output. This bit will only be set if <code>dprpGioRangedIn</code> or <code>dprpGioRangedOut</code> is set.
<code>dprpGioTimeStamp</code>	The port/channel provides a time stamp for data returned via the <code>DgioGetData</code> function. The time stamp indicates the time when the data was sampled.

The above values are used for both the port properties and the channel properties for DGIO ports. The property value for a DGIO port is the union (bitwise 'or') of the channel properties for all of the channels on the port.

The properties: `dprpGioDiscreteIn`, `dprpGioDiscreteOut`, `dprpGioRangedIn`, `dprpGioRangedOut`, `dprpEncodedIn`, `dprpEncodedOut` are mutually exclusive. A given channel will have one and only one of these properties.

## Overview of DGIO Operation

The DGIO interface provides a generic way of reading or writing data to a wide variety of sensor and user interface type devices. A DGIO port will provide one or more channels of data to or from the device. Data is read from input channels using `DgioGetData`, and written to output channels using `DgioPutData`.

There are three basic data types supported by the DGIO interface. These are: discrete, ranged, and encoded.

The discrete data type provides up to 32 bits of input or output per channel. Data for a discrete channel is always read or written as a 32 bit value. The number of meaningful data bits in the data value can be determined using the function `DgioGetDiscreteMask`. This function returns a bit mask that identifies the defined bits for the channel. Discrete input channels (`dprpDgioDiscreteIn`) are used to read binary sensors or user input devices. These are such things as switches or buttons, light threshold detectors, contact switches, etc. Discrete output channels are used to write to binary output devices, such as LEDs, relays or solenoids, etc.

The ranged data type is used for values that specify a continuous range. These values are specified as 32-bit signed integers. A ranged value channel provides up to eight axes of values. For example: a three axis accelerometer provides acceleration values in the X, Y, and Z axis directions. This could be implemented as a DGIO port providing three, single valued, ranged input channels, or as a DGIO port providing a single, triple valued, ranged input channel. The number of value axes associated with a channel can be determined using the function `DgioGetRangedCount`.

The definition of the encoded data type is reserved for future use.

Some devices require additional setup or configuration for selecting operating modes, ranges, or other operating properties of the port. The functions `DgioPutConfigValue` and `DgioGetConfigValue` are provided for this purpose. For example, an accelerometer might have multiple, selectable, sensitivity ranges. `DgioGetConfigValue` can be used to query the current sensitivity setting, and `DgioPutConfigValue` can be used to select a sensitivity setting. Refer to product documentation for information on the configuration values and their meanings for a given product.

Devices that support both input and output may be implemented in several ways. If a device has, for example, both ranged inputs and ranged outputs, it may have a single port with two channels, with the inputs on one channel and the outputs on another channel. Another possibility is that the device would implement two ports, one with a single input channel and the other with a single output channel.

Refer to the product documentation to determine the ports and channels are supported by a given device and to determine the actual meanings of the input and output values for the channels.

## DGIO API Functions

The following API functions make up the DGIO interface.

### DgioGetVersion(char \* szVersion)

*Parameters:*

szVersion - pointer to buffer to receive version string

This function returns a version number string identifying the version number of the DGIO DLL.

### DgioGetPortCount(HIF hif, INT32 \* pcprt)

*Parameters:*

hif - open interface handle on the device  
pcprt - pointer to variable to receive count of ports

This function returns the number of DGIO ports implemented by the device specified by *hif*.

### DgioGetPortProperties(HIF hif, INT32 prtReq, DWORD \* pdprp)

*Parameters:*

hif - open interface handle on the device  
prtReq - port number to query  
pdprp - pointer to variable to return port property bits

This function returns the port properties bits for the specified DGIO port. The port properties bits indicate the specific features of the DGIO port specification implemented by the specified port. The properties for a DGIO port is the union of the channel properties of the channels on the port.

### DgioEnable(HIF hif)

*Parameters:*

hif - open interface handle on the device

This function is used to enable the default DGIO port (port 0) on the specified device. This function must be called before any functions that operate on the DGIO port may be called for the specified device.

### DgioEnableEx(HIF hif, INT32 prtReq)

*Parameters:*

hif - open interface handle on the device  
prtReq - DGIO port number

This function is used to enable a specific port on devices that support multiple DGIO ports. This function must be called before any functions that operate on the DGIO port may be called. The *prtReq* parameter specifies the port number of the DGIO port to enable.

When a DGIO port is enabled, all channels in the port are initialized to their default state.

### DgioDisable(HIF hif)

*Parameters:*

hif - open interface handle on the device

This function is used to disable and end access to the currently enabled DGIO port on the specified interface handle.

### DgioGetChanCount(HIF hif, INT32 \* pcchn)

*Parameters:*

hif - open interface handle on the device  
pcchn - pointer to variable to receive count of channels

This function returns the number of channels available on the DGIO port enabled on interface handle *hif*.

### DgioGetChanProperties(HIF hif, INT32 chnReq, DWORD \* pdprp)

*Parameters:*

hif - open interface handle on the device  
chnReq - channel number to query  
pdprp - pointer to variable to return port property bits

This function returns the channel property bits for the specified DGIO channel on the port enabled on interface handle *hif*. The channel property bits indicate the specific features of the DGIO specification implemented by the channel.

### DgioGetDiscreteMask(HIF hif, INT32 chn, DWORD \* pdwMask)

*Parameters:*

hif - open interface handle on the device  
chn - channel number  
pdwMask - variable to receive bit mask

For DGIO port channels that support the *dprpGioDiscreteIn* or *dprpGioDiscreteOut* properties, this function returns the bit mask that indicates which bits in the input/output value are defined. There will be a 1 in each bit position that corresponds to a defined input/output bit for the channel.

### DgioGetRangedCount(HIF hif, INT32 chn, INT32 \* pcvalRng)

*Parameters:*

hif - open interface handle on the device  
chn - channel number  
pcvalRng - variable to receive count of values

For DGIO port channels that support the *dprpGioRangedIn* or *dprpGioRangedOut* properties, this function returns the number of values associated with the channel. The set of ranged values are analogous to axes of values. For example, an analog joystick has two axes of motion (X and Y). This device would return a count of two to indicate that each time the ranged values are obtained, there will be two values that can be returned. The maximum number of values that a ranged value channel can have is eight.

**DgioInputEnable(HIF hif, INT32 chn)***Parameters:*

hif                   - open interface handle on the device  
chn                   - channel number

This function is used to enable input on the specified channel of the DGIO port enabled on interface handle *hif*. If the channel doesn't support input, this function is ignored. Some device channels don't support enabling/disabling inputs. Input on these channels are automatically enabled when the port is enabled and disabled when the port is disabled. In this case, the *DgioInputEnable* and *DgioInputDisable* functions are ignored.

**DgioInputDisable(HIF hif, INT32 chn)***Parameters:*

hif                   - open interface handle on the device  
chn                   - channel number

This function is used to disable input on the specified channel of the DGIO port enabled on interface handle *hif*. If the channel doesn't support input, this function is ignored. Some device channels don't support enabling/disabling inputs. Input on these channels is automatically enabled when the port is enabled and disabled when the port is disabled. In this case, the *DgioInputEnable* and *DgioInputDisable* functions are ignored.

**DgioOutputEnable(HIF hif, INT32 chn)***Parameters:*

hif                   - open interface handle on the device  
chn                   - channel number

This function is used to enable output on the specified channel of the DGIO port enabled on interface handle *hif*. If the channel doesn't support output, this function is ignored. Some device channels don't support enabling/disabling the output. Output on these channels is automatically enabled when the port is enabled and disabled when the port is disabled. In this case, the *DgioOutputEnable* and *DgioOutputDisable* functions are ignored.

**DgioOutputDisable(HIF hif, INT32 chn)***Parameters:*

hif                   - open interface handle on the device  
chn                   - channel number

This function is used to disable output on the specified channel of the DGIO port enabled on interface handle *hif*. If the channel doesn't support output, this function is ignored. Some device channels don't support enabling/disabling the output. Output on these channels is automatically enabled when the port is enabled and disabled when the port is disabled. In this case, the *DgioOutputEnable* and *DgioOutputDisable* functions are ignored.

**DgioPutConfigValue(HIF hif, INT32 chn, INT32 ival, DWORD dwVal)**

*Parameters:*

- hif - open interface handle on the device
- chn - channel number
- ival - index specifying the configuration value being written
- dwVal - configuration value to write

This function is used to write configuration values to the specified channel on the DGIO port. Configuration values are used to select operating modes or other options controlling the operation of the channel. The set of configuration values supported and their specific meanings are device dependent. Refer to the device documentation for information on the configuration values for a given device.

**DgioGetConfigValue(HIF hif, DWORD chn, INT32 ival, DWORD \* pdwVal)**

*Parameters:*

- hif - open interface handle on the device
- chn - channel number
- ival - index specifying the configuration value being written
- pdwVal - variable to receive configuration value to read from the channel

This function is used to read configuration values from the specified channel on the DGIO port. The set of configuration values supported and their specific meanings are device dependent. Refer to the device documentation for information on the configuration values for a given device.

**DgioPutData(HIF hif, INT32 chn, INT32 ival, void \* pvData, DWORD cbData, BOOL fOverlap)**

*Parameters:*

- hif - open interface handle on the device
- chn - channel number
- ival - index number of the starting value to return
- pvData - pointer to the data to send to the device
- cbData - number of bytes of data being sent
- fOverlap - TRUE for overlapped i/o, FALSE for blocking i/o

This function will send the specified data to the specified channel of the DGIO port enabled on interface handle *hif*. The nature of the data to send depends on the type of DGIO port and its current operating mode. The amount of data sent must be consistent with the current operating mode of the channel. This call is only meaningful for channels that support output.

The *ival* parameter indicates the index of the starting value to put. For example, an XY output device would support two ranged output values (the X and Y axes). To write both X and Y, an index value of 0 and a byte count of 8 (2 \* cbGioRanged) is specified. To write only the X value, an index of 0 and a byte count of 4 is used. To write only the Y value, an index of 1 and a byte count of 4 is used.

Discrete output channels (channels with the `dprpGioDiscreteOut` property) only support a single output value with an index of 0. It is only meaningful to specify 0 as the index and 4 (defined as `cbGioDiscrete` in `dgio.h`) as the byte count for these channels.

**DgioGetData(HIF hif, INT32 chn, INT32 ival, void \* pvData, DWORD cbData, BOOL fOverlap)**

## Parameters:

hif	- open interface handle on the device
chn	- channel number
ival	- index number of the starting value to return
cbData	- number of bytes of data being sent
pvData	- pointer to the data to send to the device
fOverlap	- TRUE for overlapped i/o, FALSE for blocking i/o

This function will read data from the specified channel of the DGIO port enabled on interface handle *hif*. The nature of the data returned depends on the type of DGIO port and its current operating mode. The amount of data read must be consistent with the current operating mode of the port. This call is only meaningful for ports that support input.

The *ival* parameter indicates the index of the starting value to read. For example, an accelerometer device might report three ranged values (the X, Y, and Z axes). To read all three values, the index value 0 and a byte count of 12 (3 \* cbGioRanged) is specified. To read the X and Y values, but not Z, an index value of 0 and a byte count of 8 would be used. To read only the Z value, an index of 2 and a byte count of 4 would be used.

On channels that support the *dprpGioTimeStamp* property, a time stamp giving the time when the data value was sampled is returned as the last data value. For example, the three-axis accelerometer mentioned above would report the ranged value count as four if it supported time stamping. The four values would be X, Y, Z, and time. The time value is an unsigned 32 bit quantity (DWORD) giving the number of microseconds since the last time the device was powered on/reset.

Discrete input channels (channels with the *dprpGioDiscreteIn* property) without time stamping support a single data value with index of 0. Discrete input channels that support time stamping return two data values. The first is the input value and the second is the time stamp.