

Digilent Adept System Programmer's Reference Manual

Revision: September 2, 2010



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

Introduction

The Digilent Adept System is a collection of application programs, runtime libraries and drivers that allow users to interact programmatically with various Digilent products. The Digilent Adept Runtime contains the shared libraries and drivers that support the applications programs, and the Digilent Adept SDK provides the link libraries and documentation needed to write applications programs to work with the Adept System.

This document provides an overview of the operation of the Adept system and the basic concepts that need to be understood to write software to work with it. The detailed documentation for each subsystem is contained in a separate programmer's reference manual for that subsystem.

Overview

The Adept runtime is made up of a number of shared dynamic libraries that provide applications programming interfaces (APIs) used to access the various features of the system. In the Microsoft Windows version of the system, the dynamic libraries are implemented as Dynamic Link Libraries (DLL files). In the Linux version of the system, the dynamic libraries are implemented as dynamically linkable Shared Libraries (.so files).

The API functions are divided into sets of related functions, called Applications Protocol Types (APTs). Each APT provides access to a related set of communications protocols or related input/output operations. All of the functions related to a specific APT are contained in a single library for that APT. For example, all of the functions related to accessing the SPI (serial peripheral interface) communications facility are part of the DSPI APT and are contained in the DSPI library (dspil.dll on Windows and dspil.so on Linux).

There is also a device access and management facility whose functions are contained in the DMGR library (dmgr.dll or dmgr.so). The DMGR functions are used to open and close devices, enumerate devices available to the system, and query the capabilities of the devices.

The following APTs, and associated libraries, are provided by the system:

DMGR	Device access manager; Provides functions for opening, closing, enumerating, querying and setting device information, and querying which APTs are supported by specific devices.
DMGT	Device management; Provides functions for managing device level functions on Adept compatible devices. This includes such things as turning power on or off, resetting the device, querying information about power supplies on the device, and so on.
DJTG	JTAG interface; Provides functions for accessing and controlling an IEEE 1149 compatible JTAG scan chain connected to the device.

DPIO	Pin Input/Output; Provides functions for reading and writing input/output pins on the device.
DEPP	Asynchronous Parallel Interface; Provides functions for accessing parallel input/output ports on the device and reading/writing the registers in the port.
DSTM	High Speed Stream Input/Output; Provides functions for accessing a high speed parallel input/output channel.
DSPI	Serial Peripheral Interface; Provides functions for accessing SPI ports on the device and communicating with SPI devices connected to the port.
DTWI	Two Wire Serial Interface; Provides functions for accessing I2C compatible two wire serial interface busses and communicating with connected devices.
DACI	Asynchronous Communications Interface; Provides functions for accessing UART interface ports on the device and communicating with connected UART devices.
DAIO	Analog Input/Output; Provides functions for accessing analog input (A/D) and analog output (D/A) ports on the device.
DEMC	Electro-Mechanical Control; Provides functions for accessing and controlling various kinds of electro-mechanical actuator devices, such as RC servos, DC motors, and stepper motors.
DGIO	General Sensor and User Input/Output; Provides generic functions for accessing various kinds of sensor and user interface devices. This includes such sensor devices as thermometers and accelerometers and U/I devices such as buttons, switches, and LEDs.
DPCUTIL	This DLL provides backward compatibility with the earlier Adept 1.X system API functions. This allows applications written for Adept 1.X to operate with newer Adept systems.

Header Files

The Adept system contains a large number of API functions, data types, and constants. These symbols are defined in a set of header files that are part of the Adept SDK.

Declarations that are global to the entire system are contained in the header file: `dpcdecl.h`

The function prototypes and symbols definitions related to each APT are contained in an APT specific header file. These are: `dmgr.h`, `dmgt.h`, `djtg.h`, `dspi.h`, etc.

Note: The header file `dpcutil.h` is provided as part of the Adept SDK. This header file is present to enable existing projects built using the Adept 1.X SDK to be built using the Adept 2.X or later SDK. The API functions in `DPCUTIL` are deprecated and it is not recommended to write new programs using them. Documentation for the `DPCUTIL` functions can be found in the Adept 1.X SDK available on the Digilent web site.

Library Files

The implementation of the Adept system is contained in a set of dynamically linkable shared libraries.

In the Windows version of the system, there is a separate DLL, and import library for each APT. These are: dmgr.dll and dmgr.lib, dmgt.dll and dmgt.lib, etc. The appropriate import libraries for the APTs being used in a program should be included in the list of inputs to the linker when building a project that uses the Adept system. The import library is used when building the application program. The DLL must be present for the program to load and run.

In the Linux version of the system, there is a separate shared library file for each APT. These are: dmgr.so, dmgt.so, djtg.so, etc. These library files are used both to build and run a program. The shared library is included in the link list when building a program, and must be present on the target system in order for the program to run.

Device Access Errors

Unless otherwise specified, the Adept system API functions return a boolean value indicating whether the function succeeded or not. Adept API functions return TRUE if successful, and FALSE if not. When a function fails (i.e. returns FALSE), an error code indicating the nature of the failure can be obtained using the function DmgrGetLastError.

DmgrGetLastError returns the error code value of the last error that occurred in the context of the calling process and thread.

Error code values are defined in the header file dpcdecl.h, and use the defined type ERC. The following lists the defined error codes and their meanings:

ercNoErc	No error occurred
ercNotSupported	Capability or function not supported by the device
ercTransferCancelled	An overlapped data transfer operation was cancelled by the application program, or the data transfer was cancelled internally because it exceeded the time limit.
ercCapabilityConflict	The port being enabled requires resources already in use by another enabled port.
ercCapabilityNotEnabled	The port specified in the operation hasn't been enabled.
ercEppAddressTimeout	EPP Address strobe timeout
ercEppDataTimeout	EPP Data strobe timeout
ercDataSndLess	Data send failed or peripheral did not received all the data sent
ercDataRcvLess	Data receive failed or peripheral sent less data
ercDataRcvMore	Peripheral sent more data
ercDataSndLessRcvLess	Two errors: ercDataSndLess and ercDataRcvLess
ercDataSndLessRcvMore	Two errors: ercDataSndLess and ercDataSndFailRcvMore
ercInvalidPort	The port number specified isn't a valid port on the device
ercBadParameter	Function parameter out of range
ercTwiBadBatchCmd	Invalid command encountered in a TWI batch buffer
ercTwiBusBusy	Timed out waiting for TWI bus to become available
ercTwiAdrNak	TWI address not ack'd by the addressed slave device
ercTwiDataNak	TWI data not ack'd by the addressed slave device

ercTwiSmbPecError	Packet CRC error when using packet error checking
ercAlreadyOpened	Device already opened
ercInvalidHif	The specified interface handle isn't the handle of an open device
ercInvalidParameter	One or more of the parameters to the API function contained invalid or out of range values.
ercTransferPending	There is an incomplete overlapped data transfer operation in progress.
ercApiLockTimeout	API waiting on pending API timed out
ercPortConflict	The specified port is already enabled
ercConnectionFailed	Unspecified failure to connect to the device
ercControlTransferFailed	Control transfer failed
ercCmdSendFailed	A communications error occurred in sending a command to the device.
ercStsReceiveFailed	A communications error occurred while reading a result from the device.
ercInsufficientResources	Memory allocation failed, insufficient system resources
ercInvalidTFP	Internal error, DVT rejected the transfer structure sent by public API
ercInternalError	Unspecified Internal Error
ercTooManyOpenedDevices	The limit on the total number of simultaneously opened interface handles has been exceeded.
ercConfigFileError	Processing of configuration file failed
ercDeviceNotConnected	Device not connected
ercEnumNotFree	Device Enumeration failed because another enumeration is still running.
ercEnumFreeFail	Device Enumeration list could not be freed
ercInvalidDevice	OEM ID check failed
ercDeviceBusy	Device is busy

Transport Types

The Adept system provides a uniform method of access and communications with devices connected to a PC using various communications mechanisms. The transport type specifies the mechanism by which a device is connected to or communicates with the PC. This includes such things as USB and Ethernet. The type DTP is used for constants or variables specifying the transport type. Each individual transport type has a defined bit in the DTP value. A set of transport types is indicated using the bit-wise 'or' of the transport types of interest.

The following DTP values are defined:

dtpUSB	Communication with the device is done using USB
dtpEthernet	Communication with the device is done using Ethernet.
dtpParallel	Communication with the device is done using a parallel printer port.
dtpSerial	Communication with the device is done using a serial port.

Device Capabilities

The Adept system provides for a large number of APTs. Any given device will provide an implementation for one or more of the APTs. The 'device capabilities' identifies the specific set of APTs implemented by a device. The defined type DCAP is used for constants or variables specifying

device capabilities. Each APT is represented by a specific bit in a DCAP value. DCAP is a synonym for APT, and DCAP values are used to refer to sets of APTs in the Adept system API functions.

When enumerating devices, for example, it is possible to enumerate for devices that support a specific set of capabilities. In this case, the DCAP values passed to the enumeration function would contain the bit-wise 'or' of the DCAP values for the APTs of interest.

Alternatively, given a device, it is possible to determine all of the capabilities (APTs) supported by that device. The DCAP value returned by the device contains the bit-wise 'or' of the bit values for all APTs supported by the device. Having a DCAP bit set in the returned capabilities value means that the device implements at least one port for the corresponding APT.

The following DCAP values are defined:

dcapJtg	IEEE 1149 compatible JTAG port (DJTG.DLL)
dcapPio	Pin Input/Output port (DPIO.DLL)
dcapEpp	Asynchronous parallel interface port (DEPP.DLL)
dcapStm	High speed streaming parallel interface port (DSTM.DLL)
dcapSpi	Serial Peripheral Interface port (DSPI.DLL)
dcapTwi	Two Wire Serial Interface port (DTWI.DLL)
dcapAci	Asynchronous Communications Interface port (DACI.DLL)
dcapAio	Analog Input/Output port (DAIO.DLL)
dcapEmc	Electro-Mechanical Control port (DEMC.DLL)
dcapGio	General Sensor and User Interface port (DGIO.DLL)

Additionally, the value *dcapAll* is defined, which has all bits set. This can be used when enumerating devices to specify that devices implementing any APT should be returned.

Interface Handles

An interface handle is a token that represents an active connection to a specific device. An interface handle is obtained using either the *DmgrOpen* or *DmgrOpenEx* functions. In general no access can be made on a device until a valid interface handle is obtained for the device, and the interface handle provides a context for access to the device.

The defined type *HIF* is used for interface handles.

An interface handle is released by calling *DmgrClose*, and interface handles should be closed when access to the device is no longer required.

There can be multiple simultaneously opened handles outstanding against a given device in the system. These handles can be held by the same process or different processes. A process can hold multiple open handles for the same or different devices, and multiple processes can hold simultaneously open handles for the same device. For some transport types, such as Ethernet, interface handles may be simultaneously held by processes running on different computers. Interface handles are not inherited by child processes.

Although any number of interface handles can be open simultaneously for access to a given device, only a single operation can be in progress on a given device at a time. If multiple processes or threads attempt to simultaneously access a device, the system will serialize access to the device so that only

a single operation occurs at a time. When a process or thread attempts to perform an operation on a device that is currently busy, it will block until the first operation has completed.

Some I/O operations can take a considerable time to complete, and so, the blocked thread may be blocked for a considerable time. There is a master timer that keeps track of how long an I/O operation has taken to complete and aborts the operation if it has exceeded the time limit. This is to prevent a malfunctioning process from locking out access to a device. The default value for this master timeout is 10 seconds, but can be set to different values.

Ports

A port is an instantiation of the hardware required to support the functionality of an APT (or some subset of the APT) on a given device. For example, a device might support two SPI ports. This means that the device contains two sets of the hardware required to implement an SPI interface and that the two ports can be accessed simultaneously and independently. It is possible to query the number of ports the device supports for each supported APT.

Port numbers are small integers starting with 0. Each APT provides its own set of port numbers. So, for example, a device that provided three SPI ports and one JTAG port would have SPI ports numbered 0, 1, and 2, and a JTAG port numbered 0.

A port must be enabled before it can be used and disabled when it is no longer needed. When a port is enabled, the device ensures that the hardware resources used by the port have been correctly configured and initialized for access. When a port is disabled, the device performs actions to release the hardware and discontinue its use. The hardware state of a port is not preserved after it is disabled, and no assumptions can be made about the state of I/O pins associated with the port after it has been disabled. In some cases, disabling a port may mean that portions of the device are powered down or in other ways affected which invalidate the state of the port. Some ports on some devices may preserve pin state after a port has been disabled, but this behavior cannot be relied upon and may change with different versions of the device.

Only one port can be enabled at a time on an interface handle. An application could serially access multiple ports on a handle by enabling and disabling the ports as necessary. However, an application that requires simultaneous access to multiple ports should open as many interface handles as needed to allow the required number of ports to be simultaneously enabled.

In some cases, hardware resources may be shared between ports of two or more different APTs. In this case only one of these ports can be enabled at a time. For example, SPI is a synchronous serial communications protocol, and JTAG is also a synchronous serial protocol. A given device could have one set of serial communications hardware able to support either SPI or JTAG. This device could report that it supports one SPI port and one JTAG port. However, since there is only a single set of the hardware used by these ports, only one of them could be enabled at a time. In this case, if the SPI port is enabled, an attempt to enable the JTAG port will fail. The SPI port would need to be disabled before the JTAG port could be enabled.

In some cases, the device may not contain all of the hardware required to support a particular port. The implementation of the port may depend on an optional hardware module being attached to a particular connector on the device. For example, a DEMC (Electro-Mechanical Control) DC motor port might require that an h-bridge module be attached to a connector on the device to provide the circuitry

required to actually drive the motor. In the case of Adept compatible devices containing programmable logic (such as FPGA board), it may be necessary for supporting logic to be configured into the programmable logic device. The Adept compatible device may not be able to determine whether the external hardware is connected (or configured). In this case, enabling the port might succeed, but the port still not function correctly because the required external hardware isn't present.

Port Properties

Most APTs define a core set of functionality plus optional features that any given device may or may not support. The port property value for a given port identifies which optional features of the APT are supported by that port.

The defined type DPRP is used for constants and variables related to port properties. The port property values for each APT are defined in the header file associated with the APT. Each defined set of optional functionality for a given APT has a DPRP bit defined.

Refer to the programmer's reference manuals for each APT for documentation on the port property values defined for each APT.

Channels

Some APTs are defined such that a given port can have multiple data streams or control streams associated with it. In this case, a channel number is used to refer to a specific data/control stream associated with the port. For example, the DEMC (Electro-Mechanical Control) APT defines functions for controlling RC servos. A given DEMC servo port can control up to 32 servos from a single port. A servo port, therefore, could define up to 32 individual servo channels for each servo port. Most APTs that have ports supporting multiple channels provide functions that allow querying the number of channels available on each port. Alternatively, refer to product documentation for the specific product.

Channel numbers are small integers starting with 0.

Product Identifier

The product identifier is a binary value stored in each Adept compatible device that can be used to uniquely identify the specific type of device. The defined type PDID is used for product identifiers.

The PDID is a 32-bit unsigned value that contains a 12-bit product family field, a 12-bit product variant field, and an 8-bit firmware type identifier field. The product family value identifies the type of product. If there are multiple versions of the product available, the product variant value identifies the product version. The firmware identifier value is used internally by the Adept system to associate the correct firmware image with the device for devices that support dynamically reconfigurable firmware.

For example, the Nexys 2 is a Digilent field programmable gate array (FPGA) board. The product family value for the Nexys 2 is 0x006. This board is available with either a 500K gate die or a 1200K gate die. The product variant value for the 500K gate version is 0x001 and the product variant value for the 1200K gate version is 0x002.

The following macros are provided in *dpcdel.h* for access to the fields of the PDID:

ProductFromPdid(pdid) Return the product family from the PDID value

VariantFromPdid(pdid)	Return the product variant from the PDID value
FwidFromPdid(pdid)	Return the firmware type identifier from the PDID value

Device Attribute Information

Each Adept compatible device contains a number of attributes that can be queried, and in some cases, set. Some of these attributes are universal to all Adept compatible devices, and some are specific to certain kinds of devices.

These attributes are queried using the `DmgrGetInfo` function, and for those that are settable, they are set using the `DmgrSetInfo` function. The specific attribute is specified using a DINFO value. The DINFO type and the available DINFO values are defined in the header file `dpcdecl.h`.

The following list gives the queryable and settable DINFO values and their meanings:

dinfoUsrName	A user settable name string stored in the device that can be used to uniquely identify the device. The user name string can be used as a connection string to open the device. Each device comes from the factory with a default user name value which can be set to a different value. This can be used to differentiate between devices if more than one of the same kind of device are connected to a system.
dinfoProdName	A descriptive string that can be used to identify the kind of device. Each different kind of Adept compatible device has a unique product name string. This string is not settable.
dinfoPDID	A product identifier value. The product identifier is a binary value that specifies a product family, a product variant within the product family and the firmware type used by the device. Software that adjusts its behavior depending on the device it is working with can use this value to identify the specific Adept device being used.
dinfoProdID	The product family identifier from the PDID value.
dinfoSN	The device serial number. The serial number is a 12 digit alpha-numeric string prefixed with the characters 'SN:' (e.g. SN:210174228096). All Adept compatible devices have a unique serial number value. This can be used to distinguish between different similar devices. Digilent devices have the last six digits of the serial number string printed on a bar-code label attached to the device, or in some cases the device packaging, if the device itself isn't large enough to hold the label.
dinfoDCAP	The device capabilities value. This identifies the capabilities (supported APTs) implemented by the device.
dinfoFWVER	The firmware version number of the device firmware.
dinfoIP	The IP (Internet Protocol) address for network devices. This value is only defined for network devices and is the IP address assigned to the

device. In some cases, this value is settable. The IP address is an alpha-numeric string prefixed with the characters 'IP:' and the actual IP address, (e.g. IP:192.168.2.13).

dinfoMAC

The Ethernet MAC address for Ethernet devices. This value is only defined for Ethernet connected devices. Devices that have a MAC address use the MAC address as the serial number. This is an alpha-numeric string with the device address prefixed with the characters 'MAC:'.

Device Enumeration

Enumeration is the process of identifying the Adept compatible devices connected to the user's computer or known to the user's computer. The Adept system supports dynamically discovering what devices are currently accessible from the user's computer, as well as predefining a list of devices known to the system. The enumeration support allows filtering the returned list of devices by communications type (DTP) and by supported capabilities (DCAP). The enumeration support functions are part of the DMGR subsystem and are contained in the DMGR.DLL library.

Enumeration can be performed as a blocking operation (i.e. a function that doesn't return until the enumeration has been completed) by calling either `DmgrEnumDevices` or `DmgrEnumDevicesEx`. Enumeration can also be performed as a non-blocking process by calling `DmgrStartEnum`. This function returns immediately and the enumeration is performed by a separate, concurrent thread maintained by the Adept system.

While performing a non-blocking enumeration, it is possible to determine the number of devices discovered so far and query information about them while the enumeration is progressing. The function `DmgrIsEnumFinished` can be used to determine when the enumeration has completed. If necessary, the function `DmgrStopEnum` can be used to end a non-blocking enumeration before it has completed.

The result of an enumeration is a list of the devices that satisfy the specified filter constraints. This list is maintained in a data structure internal to the Adept system. There can only be a single instance of this enumeration list per process. Once access to the results of an enumeration is no longer required, it must be released (by calling `DmgrFreeDvcEnum`) before another enumeration can be performed.

The function `DmgrGetEnumCount` is used to determine the number of devices in the enumeration. Information about a specific device in the enumeration is obtained by calling `DmgrGetDvc`. The caller provides a DVC structure that will be populated with information about the device.

The amount of time required to perform an enumeration depends on the transport type of the devices being discovered. Devices that are directly connected to the user's computer (e.g. USB devices) can be discovered quickly. In some cases, discovery of devices that are indirectly connected to the computer (e.g. Ethernet devices), can be very slow. Some transport types (e.g. parallel port devices) can't be discovered at all. The `DmgrEnumDevices` function enumerates devices of all transport types that can be discovered quickly. This function will always return quickly (i.e. within 10's to 100's of milliseconds). It does not enumerate devices of transport types that can't be discovered quickly. `DmgrEnumDevices` also examines the device table and returns devices found in the device table.

The DmgrEnumDevicesEx and the DmgrStartEnum functions allow filtering the devices returned by specifying the device transport types (DTP) to be enumerated and device attribute (DINFO) values. This function must be used to discover devices with transport types that can't be discovered quickly. This function also allows filtering devices from the device table and discovered devices separately.

When using the DmgrEnumDevicesEx or DmgrStartEnum functions to filter based on device attributes, the following attributes can be specified for the attribute filter:

dinfoNone	don't perform device attribute filtering
dinfoUserName	return devices matching this user name
dinfoProdName	return devices matching this product name
dinfoPDID	return devices matching this PDID value
dinfoSN	return devices matching this serial number
dinfoDCAP	return devices matching this DCAP value. This means only return devices if they have at least one port for each APT specified by the DCAP value.

Device Table

The device table is used to maintain a list of devices known to the Adept system whether they are connected to the user's computer and discoverable or not. Use of the device table is optional in most cases.

The device table provides the ability to associate an alias with a connection string to a device. This alias can be used as a connection string when opening a device. When the connection string passed to the DmgrOpen or DmgrOpenEx functions is in the form of a name, the device table is examined to see if the specified name matches an alias in the table. If a match is found, the connection string associated with the alias is used to make the connection to the device.

In general, the use of the device table is optional. However there are cases where its use is required:

Backward compatibility for applications written to use the earlier Adept version 1 API functions requires use of the device table. Version 1 of the Adept system only allowed connection via alias strings stored in the device table. In this case, it will be necessary to create entries in the device table to enable access to Adept compatible devices by Adept version 1 applications.

Devices that are not discoverable require entries in the device table. An example of a device that is not discoverable is an ethernet connected device that is not on the local subnet. The protocol that the Adept system uses to discover ethernet devices is not routable, and therefore ethernet devices that are accessed through a gateway or router can only be accessed by having the appropriate connection string associated with an alias in the device table.

The Adept system provides API functions for device table management as part of the DMGR subsystem implemented in DMGR.DLL. In addition to the functions that allow manipulation of the device table programmatically, the Windows version of the system provides a function DmgrOpenDvcMg that displays a modal dialog box providing a user interface for device table management.

Device Connection String

A device connection string is the string that is passed to either `DmgrOpen` or `DmgrOpenEx` to specify the device to be opened. This connection string can be one of several types of strings:

Name string: The user name is a user settable, alpha-numeric name string stored in the device that can be used as a connection string. An alias is an alpha-numeric name string stored in the device table. When a name string is specified as the connection string, the device table is examined to find an entry matching the specified name. If a match is found in the device table, the associated connection string is used. If a match is not found in the device table, an enumeration of discoverable devices is performed. The result of the enumeration is examined for a device whose user name matches the specified name string.

Serial number: The device serial number can be used as a connection string. The serial number is specified using a string with the prefix 'SN:' followed by the 12 digit alpha-numeric serial number string for the device. When a serial number string is used, an enumeration of discoverable devices is performed, and the result examined for the device whose serial number matches the request.

Network address: For network devices the IP address or the MAC address can be used. These are specified using the prefix 'IP:' followed by the IP address, or the prefix 'MAC:' followed by the MAC address.

Overlapped Input/Output

In operating environments that support it, the Adept runtime system allows for overlapped I/O on some function calls. All functions that support overlapped I/O have an "fOverlapped" parameter as the last parameter in the parameter list. If a non-zero (TRUE) value is passed for fOverlapped, the function call will be performed as overlapped I/O. In this case, an I/O transaction will be scheduled to be performed, and the Adept function will return immediately.

The status of an overlapped I/O operation can be queried by calling the function `DmgrGetTransResult`. This function will return an error with the error code `ercTransPending` if the overlapped I/O operation has not yet completed. It will return an error with some other error code if the operation has failed.

The system supports only a single pending overlapped operation to be outstanding at one time on a given interface handle (HIF). Any number of simultaneous operations can be pending on multiple handles, but only one operation can be in process on any given handle. An attempt to perform a new I/O operation (overlapped or not) on a handle with a pending overlapped operation will fail and return an error. In addition, only a single operation can be in progress on any given device, whether overlapped or not. If an overlapped operation is in progress on a device, any new operations on that device will block until the overlapped operation has completed.