CHAPTER 2

CDIL Interface

The Communications Desktop Integration Library (CDIL) is a small library for Windows and Macintosh applications that need to communicate with Newton OS devices. Operations are provided via a C language application program interface (API). The data exchanged is free-form. That is, the CDIL does not impose any sort of data format, nor does it imply any high, or application, level protocols. All it provides is a stream-based communications API for sending data to and receiving data from a Newton device. This API turns around and works with transport specific APIs (such as ADSP, TCP/IP, and MNP) to transfer the data.

About the CDIL

The CDIL provides a pipe between a Windows or Macintosh desktop computer and a Newton device. The CDIL uses the passive listener model. In this model the desktop application sets up the pipe, and waits for the Newton device to initiate a connection. Once the connection is set up, both sides can read and write bytes through the pipe, and terminate the connection.

CDIL Compatibility

The following changes have been made:

kCDIL_Uninitialized
: This state previously identified a pipe that had been created but had never been used. Now it identifies a NULL (non-created) pipe.

kCDIL_InvalidConnection
: This state no longer exists.

kCDIL_Startup
: This state no longer exists.

kCDIL_Listening
: This state is unchanged.

kCDIL_ConnectPending
: This state is unchanged.

kCDIL_Connected
: This state is unchanged.

kCDIL_Busy
: This state no longer exists.

kCDIL_Aborting
: This state no longer exists.

kCDIL_Disconnected
: This state used to identify a pipe that used to be connected, but is no longer. Now it identifies any unconnected pipe, even if it's never been used before.

kCDIL_Userstate
: This state no longer exists.

CommErr
: Replaced by DIL_Error.

CDInitCDIL
: Renamed to CD_Startup.

CDDisposeCDIL
: Renamed to CD_Shutdown.

CDCreateCDILObject
: Replaced by CD_CreateXXX suite of functions.

CDDisposeCDILObject
: Renamed to CD_Dispose.

CDPipeInit
: Replaced by CD_CreateXXX suite of functions.

CDPipeDisconnect
: Renamed to CD_Disconnect.

CDPipeListen
: Renamed to CD_StartListening. Function returns immediately; there is no asynchronous operation, so the

	timeout, *completionHook*, and *refCon* parameters have been removed.
CDPipeAccept	Renamed to CD_Accept.
CDPipeAbort	Removed.
CDPipeRead	Renamed to CD_Read. The *eom*, *swapSize*, *destEncoding*, *completionHook*, and *refCon* parameters represent functionality that is no longer available and have been removed.
CDBytesInPipe	Renamed to CD_BytesAvailable. Only the reporting of bytes in the input buffer is supported, so the direction parameter has been removed.
CDPipeWrite	Renamed to CD_Write. The *eom*, *swapSize*, *destEncoding*, *completionHook*, and *refCon* parameters represent

	functionality that is no longer available and have been removed.
CDIdle	Renamed to CD_Idle.
CDGetPipeState	Renamed to CD_GetState.
CDSetPipeState	Removed. User states are no longer supported.
CDEncryptFunction	Removed. Encryption is no longer supported at the CDIL level.
CDDecryptFunction	Removed. Encryption is no longer supported at the CDIL level.
CDGetConfigStr	Removed. Configuration parameters are no longer necessarily specified via a configuration string.
CDGetPortStr	Removed. Configuration parameters are no longer necessarily specified via a configuration string.
CDGetTimeout	Removed. Timeout values are no longer pipe state variables.
CDSetApplication	Removed. The CDIL no longer needs the application's HINSTANCE handle.
CDFlush	Removed. This functionality could not be guaranteed for all transport services.
CDPad	Removed. This functionality was not required at the CDIL level.
CDSetPadState	Removed. This functionality was not required at the CDIL level.

Using the CDIL

Creating a CDIL Session

You must bracket all calls to CDIL functions between calls to CD_Startup and CD_Shutdown. You create a session by passing a pointer to a pipe object, a CD_Handle *, to one of the pipe creation functions: CD_CreateADSP, CD_CreateMNPSerial, CD_CreateTCP, and CD_CreateCTB. These functions create a

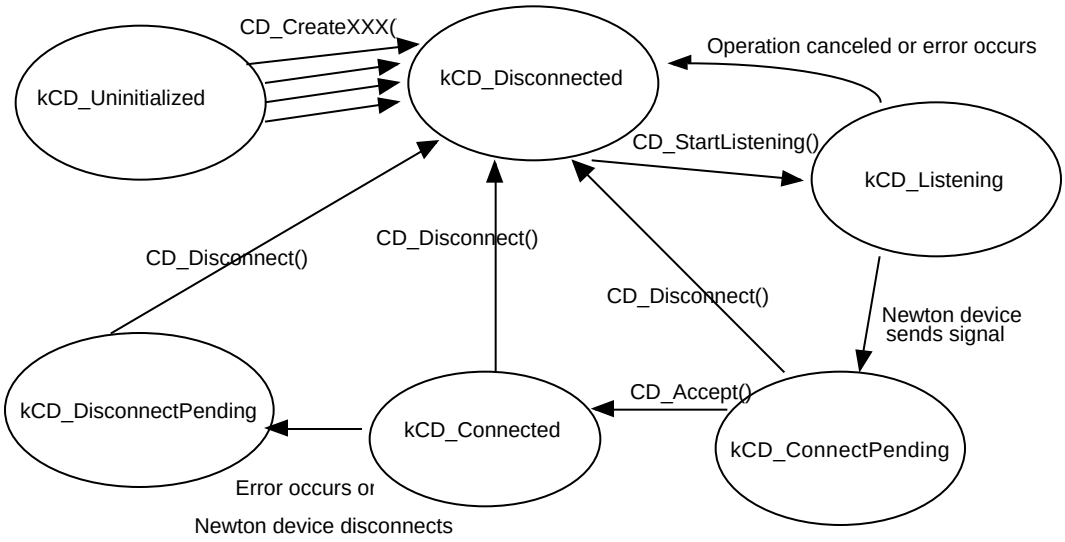
connection with the Newton device using the requested communication service: AppleTalk, MNP Serial, TCP, or a Macintosh communication tool, respectively. All of these connection services are available in Mac OS, only the TCP and MNP Serial option are available in Windows.

Each pipe creation function has an associated checking function: CD_CheckADSP, CD_CheckMNPSerial, CD_CheckTCP, and CD_CheckCTB. These functions test to see if the appropriate service is available. They are however only an indication of whether the respective pipe creation function will succeed. They check to see if the library can be loaded and initialized, but you cannot determine if the connection attempt will succeed until you try. These functions allow an application to build a dynamic set of connection options for the user to choose from.

Once you have created a pipe to the Newton device, it is in the kCD_Disconnected state. You must set it to listening mode with the CD_StartListening function, moving it to the kCD_Listening state. Once in this mode, the pipe listens for a connection request from a Newton OS device. When such a request is registered, the pipe is placed in the kCD_ConnectionPending state by the CDIL. You should detect this condition, and call CD_Accept to establish the connection. CD_Accept puts the pipe in the kCD_Connected state. In this state, you can read and write bytes through the pipe.

The CDIL can thus be seen as a finite state machine. The state diagram is shown in Figure 2-1. You can use the CD_GetState function to determine the current state of the pipe.

Figure 2-1 CDIL state diagram



Terminating a CDIL Session

You can terminate a CDIL session at any time by calling `CD_Disconnect`. This places the pipe in the `kCD_Disconnected` state, and you are then able to attempt to start a new session by calling `CD_StartListening`. The connection can also be broken by the Newton device at any point, or fail due to some sort of error. If the connection is broken by the Newton device, or due to error, the pipe is also placed in the `kCD_Disconnected` state, unless it is presently in the `kCD_Connected` state. In that case, the pipe is placed in the `kCD_DisconectPending` state where you can still read any buffered bytes.

When you no longer need the pipe object call `CD_Dispose` to free any allocated resources.

Reading and Writing Through the Pipe

You can write to a pipe with the CD_Write function so long as it is in the kCD_Connected state. The data is then buffered for you, and sent down the pipe at the next call to either CD_FlushOutput, CD_Idle, CD_Read, CD_Disconnect, or CD_BytesAvailable. The call to CD_Write can timeout. Each pipe has an associated timeout period, which defaults to 30 seconds. If the data cannot be sent within that period, CD_Write returns a kCD_Timeout error. You can set the timeout period with CD_SetTimeout; timeouts are set on a per pipe basis.

You can read data from the pipe with the CD_Read function so long the pipe is in either the kCD_Connected or kCD_DisconnectPending states. You specify how many bytes you wish CD_Read to get. CD_Read then blocks until that many bytes are available, or the call to CD_Read times out. Using the CD_BytesAvailable function, you can determined whether CD_Read would block.

As data is received from the Newton device, it is buffered by the CDIL at every possibility. You can explicitly allow the CDIL to buffer data by calling CD_Idle. This buffering is also performed by other CDIL functions, such as CD_BytesAvailable, since they present the CDIL with the opportunity to buffer this data. The underlying communication service may have a fixed-size buffer, in which case, if the Newton sends data too much data, the data could be lost. For this reason, you should call CD_Idle frequently to allow the CDIL to buffer the data.

Error Handling

Almost all CDIL functions return an error code. The code kDIL_NoError, which equals 0, indicates success. The functions descriptions in “CDIL Reference” (page 2-9) list the error codes each particular function could return if an error occurs. If an error occurs in one of the transports that the CDIL uses to implement its pipe, TCP for example, the kCD_PlatformError code is returned. In this case, you can call CD_GetPlatformError to retrieve the error code returned by the particular transport.

Code Example

The code in Listing 2-1 shows the skeleton of a CDIL session, without error checking.

Listing 2-1 A CDIL code example

```
CD_Handle  pipe;
CD_State   state;
char       dataBuffer[256];
long       count;

CD_Startup();                          // Initialize the library
CD_CreateADSP(&pipe, NULL, NULL);      // Create a connection object
CD_StartListening(pipe);               // Have that object listen for a
                                       // connection from a Newton device

while (CD_GetState(pipe) == kCD_Listening) //Wait for a connect request
{
    // If you are displaying a dialog box telling the user to
    // initiate a connection from a Newton OS device, you could
    // check for clicks on a Cancel button here.
}
if (CD_GetState(pipe) == kCD_ConnectPending)
{
    CD_Accept(pipe); // Accept the connect request
    MyFnToGetDataToSend(dataBuffer);
    CD_Write(pipe, dataBuffer, sizeof(dataBuffer));
    // This step is optional. We'd execute it if we wanted to
    // ensure that there were 100 bytes available before calling
    // CD_Read, which would otherwise block.
    do
    {
        CD_Idle(pipe);
        CD_BytesAvailable(pipe, &count);
        // You could check for clicks on menus or buttons, or call
        // WaitNextEvent here.
    }
    while (count<100);
    CD_Read(pipe, dataBuffer, 100); // Assumes we expect 100 bytes back
    CD_Disconnect(pipe); // Break the connection.
}
CD_Dispose(pipe); // Delete the pipe object
CD_Shutdown();  // Close the library
```

CDIL Reference

Type Definitions

CD_Handle — A pipe object.

Constants

CDIL States

These values are returned by CD_GetState:

kCD_Uninitialized — The pipe has not been initialized.

kCD_Disconnected — The pipe has just been created, or it has been passed to CD_Disconnect.

kCD_Listening — The pipe is listening for a connection request from a Newton device.

kCD_ConnectPending — The pipe has received a connection request from a Newton device.

kCD_Connected — The pipe is fully connected to a Newton device and can be used for data exchange.

kCD_DisconnectPending — The connection has been broken on the other end. Either the Newton device has disconnected, or a communications or network error has occurred. In this state, any buffered data can still be retrieved. The buffered data will be flushed when you call CD_Disconnect.

Timeout Intervals

These values can be used for the *timeoutInSecs* parameter to CD_SetTimout:

kCD_DefaultTimeout	The default timeout period of 30 seconds.
kCD_NoTimeout	No timeout; calls to CD_Read and CD_Write do not timeout.

Error Codes

```
kDIL_NoError                        (0)
kDIL_ErrorBase                      (-98000)
kDIL_OutOfMemory                    (kDIL_ErrorBase - 1)
kDIL_InvalidParameter               (kDIL_ErrorBase - 2)
kDIL_InternalError                  (kDIL_ErrorBase - 3)
kDIL_ErrorReadingFromPipe           (kDIL_ErrorBase - 4)
kDIL_ErrorWritingToPipe             (kDIL_ErrorBase - 5)
kDIL_InvalidHandle                  (kDIL_ErrorBase - 6)

kCD_ErrorBase                       (kDIL_ErrorBase - 200)

kCD_CDILNotInitialized              (kCD_ErrorBase - 1)
kCD_ServiceNotSupported             (kCD_ErrorBase - 2)
kCD_BadPipeState                    (kCD_ErrorBase - 3)
kCD_Timeout                         (kCD_ErrorBase - 4)
kCD_PipeDisconnected                (kCD_ErrorBase - 5)
kCD_IndexOutOfRange                 (kCD_ErrorBase - 6)
kCD_BufferTooSmall                  (kCD_ErrorBase - 7)
kCD_PlatformError                   (kCD_ErrorBase - 8)

/* Windows-specific error codes */
kCD_TCPCantFindLibraryFns           (kCD_ErrorBase - 20)
kCD_TCPInsufficientVersion          (kCD_ErrorBase - 21)
kCD_TCPNoSockets                    (kCD_ErrorBase - 22)
```

Functions

CD_Startup

```
DIL_Error CD_Startup()
```

Initializes the CDIL library.

return value An error code.

DISCUSSION

This call makes sure that any low-level transport layers (for example: ADSP, TCP/IP, MNP) are available and properly initialized. If none are available or none can be initialized, this function returns an error.

This function is usually called once at the start of your program. However, you can call it as many times as you want as long as you call `CD_Shutdown` an equal number of times.

ERROR CODES

```
kCD_PlatformError
kCD_OutOfMemory
```

CD_Shutdown

```
DIL_Error CD_Shutdown()
```

Closes any transport layers opened and initialized in `CD_Startup`, and closes and disposes of all open pipes.

return value An error code.

DISCUSSION

This function must be called once for every time you called `CD_Startup`. Usually, you just call it once at the end of your program. However, you can call it as many times as you want, as long as you don't call it more times that you've called `CD_Startup`. If this is the last call to `CD_Shutdown`, then all memory allocated by the CDIL since `CD_Startup` was called is deallocated.

ERROR CODES

kCD_CDILNotInitialized
kCD_PlatformError

CD_CheckADSP

DIL_Error CD_CheckADSP()

Determines whether the ADSP service is available.

return value An error code.

DISCUSSION

This function provides an indication of whether a call to CD_CreateADSP will succeed or fail.

ERROR CODES

kCD_CDILNotInitialized
kCD_PlatformError
kCD_ServiceNotSupported

CD_CheckCTB

DIL_Error CD_CheckCTB(const char* *toolName*)

Determines whether the CTB service is available.

toolName The name of the tool as a C string.

return value An error code.

DISCUSSION

This function provides an indication of whether a call to CD_CreateCTB will succeed or fail.

ERROR CODES

kCD_CDILNotInitialized
kCD_PlatformError
kCD_ServiceNotSupported
kDIL_InvalidParameter

CD_CheckMNPSerial

```
DIL_Error CD_CheckMNPSerial()
```

Determines whether the MNP service is available.

return value An error code.

DISCUSSION

This function provides an indication of whether a call to `CD_CreateMNPSerial` will succeed or fail.

ERROR CODES

```
kCD_CDILNotInitialized
kCD_PlatformError
kCD_ServiceNotSupported
```

CD_CheckTCP

```
DIL_Error CD_CheckTCP()
```

Determines whether the TCP service is available.

return value An error code.

DISCUSSION

This function provides an indication of whether a call to `CD_CreateADSP` will succeed or fail.

ERROR CODES

```
kCD_CDILNotInitialized
kCD_PlatformError
kCD_ServiceNotSupported
kCD_TCPCantFindLibraryFns // these bottom three are only returned
kCD_TCPInsufficientVersion // in the Windows version
kCD_TCPNoSockets
```

CD_GetSerialPortName

DIL_Error CD_GetSerialPortName(long *index*, char* *buffer*, long* *bufLen*)

Returns a user-displayable C string containing the name of a selectable serial port.

index	A zero-based value indicating the port for which you want a string.
buffer	Where to store the C string. You may pass in NULL if you only want to query for the length of the string.
bufLen	When you call this function *bufLen should contain the number of bytes of the empty buffer. CD_GetSerialPortName sets this value to the number of bytes of the string that holds the port name, including the NULL terminator.
return value	An error code.

DISCUSSION

Normal usage of this function is to start with zero, incrementing *index*, until the function returns kCD_IndexOutOfRange.

ERROR CODES

kCD_CDILNotInitialized
kCD_PlatformError
kCD_IndexOutOfRange
kCD_BufferTooSmall
KDIL_InvalidParameter

CD_CreateADSP

DIL_Error CD_CreateADSP (CD_Handle* *pipe*, const char* *name*, const char* *type*)

Creates an ADSP-based communications pipe.

pipe	Where to store the new pipe.
name	The name of the ADSP connection. This string is what appears in the Chooser list on the Newton OS device. If

	you pass NULL for this parameter, the CDIL uses a default name based on your desktop computer's preferences (for instance, on a Macintosh, it will use the strings specified in the File Sharing control panel).
type	The connection type. This is searched for by the Chooser on the Newton OS device. If you pass NULL for this parameter, the CDIL uses the type specified by the Connection / Dock application.
return value	An error code.

SPECIAL CONSIDERATIONS

ADSP pipes are only available on the Mac OS platform.

ERROR CODES

kCD_CDILNotInitialized
kCD_PlatformError
kCD_ServiceNotSupported
kDIL_InvalidParameter
kDIL_OutOfMemory

CD_CreateCTB

DIL_Error CD_CreateCTB (CD_Handle* *pipe*, const char* *toolName*, const char* *configString*)

Creates a Macintosh Communication Toolbox-based communications pipe.

pipe	Where to store the new pipe.
toolName	The name of the communication tool.
configString	A tool-dependent configuration string.
return value	An error code.

SPECIAL CONSIDERATIONS

Comm toolbox based pipes are only available on the Mac OS platform. See the Comm Toolbox documentation for valid configuration strings.

ERROR CODES

kCD_CDILNotInitialized
kCD_PlatformError
kCD_ServiceNotSupported
kDIL_InvalidParameter
kDIL_OutOfMemory

CD_CreateMNPSerial

DIL_Error CD_CreateMNPSerial(CD_Handle* *pipe*, long *port*, long *baud*);

Creates a serial communications pipe based on the MNP protocol.

pipe	Where to store the new pipe.
port	The serial port to use.
baud	The baud rate to communicate at in bytes per second. Possible values are listed in Table 2-1.
return value	An error code.

DISCUSSION

MNP is a packet-based protocol that ensures delivery of your data using compression and error correction.

Table 2-1 Possible baud rates for MNP serial connection

Windows					
	110	300	600	1200	2400
	4800	9600	14400	19200	38400
	5600	57600	115200	128000	256000
Macintosh					
	110	300	1200	2400	4800
	9600	19200	38400	57600	

Note

Not all of these baud rates are compatible with current Newton OS devices. They merely represent what is possible on the desktop platform. ◆

ERROR CODES

```
kCD_CDILNotInitialized
kCD_PlatformError
kCD_ServiceNotSupported
kDIL_InvalidParameter
kDIL_OutOfMemory
```

CD_CreateTCP

`DIL_Error CD_CreateTCP(CD_Handle* pipe, long port)`

Creates a TCP-based communications pipe.

pipe	Where to store the new pipe.
port	The TCP port to listen on. Note that once the connection is made, data transfer actually occurs on a different, randomly chosen, port. This frees up the port specified in this parameter for future connections.
return value	An error code.

ERROR CODES

```
kCD_CDILNotInitialized
kCD_PlatformError
kDIL_InvalidParameter
kDIL_OutOfMemory
kCD_TCPCantFindLibraryFns // these bottom three are only returned
kCD_TCPInsufficientVersion // in the Windows version
kCD_TCPNoSockets
```

CD_Dispose

DIL_Error CD_Dispose(CD_Handle *pipe*)

Disposes of a communications pipe created by CD_CreateADSP, CD_CreateMNPSerial, CD_CreateCTB, or CD_CreateTCP.

pipe	The pipe to dispose of.
return value	An error code.

DISCUSSION

The pipe passed to CD_Dispose can be in any state. If appropriate, the pipe is disconnected or removed from a listening state before it is deleted.

After this call, the reference to the pipe is invalid and should no longer be used.

ERROR CODES

kCD_CDILNotInitialized
kCD_PlatformError
kDIL_InvalidParameter
kDIL_InvalidHandle

CD_Disconnect

DIL_Error CD_Disconnect(CD_Handle *pipe*)

Puts the specified pipe in the kCD_Disconnected state.

pipe	The pipe to disconnect.
return value	An error code.

DISCUSSION

If the pipe is listening, it stops listening. If the pipe is connected, it is disconnected. In all cases, the state of the pipe after making this call is kCD_Disconnected. Any internally buffered data is flushed and can no longer be read with CD_Read.

ERROR CODES

kCD_CDILNotInitialized

kCD_PlatformError
kCD_BadPipeState
kCD_TimeOut
kCD_PipeDisconnected
kDIL_InvalidParameter
kDIL_InvalidHandle

CD_StartListening

DIL_Error CD_StartListening(CD_Handle *pipe*)

Makes the pipe start listening for a connection for a Newton device.

pipe	The pipe to start listening.
return value	An error code.

DISCUSSION

After the successful completion of this call, the pipe is put in the kCD_Listening state.

ERROR CODES

kCD_CDILNotInitialized
kCD_PlatformError
kCD_BadPipeState
kCD_TimeOut
kCD_PipeDisconnected
kDIL_InvalidParameter
kDIL_InvalidHandle

CD_Accept

DIL_Error CD_Accept(CD_Handle *pipe*)

Makes the pipe accept a pending connection.

pipe	The pipe to accept the connection on. This pipe should be in the kCD_ConnectPending state.
return value	An error code.

DISCUSSION

After the successful completion of this call, the pipe is fully connected, its state will be kCD_Connected, and it can be used to exchange data with a Newton OS application.

ERROR CODES

kCD_CDILNotInitialized
kCD_PlatformError
kCD_BadPipeState
kCD_TimeOut
kCD_PipeDisconnected
kDIL_InvalidParameter
kDIL_InvalidHandle

CD_Read

DIL_Error CD_Read(CD_Handle *pipe*, void* *p*, long *count*)

Reads bytes from a pipe.

pipe	The pipe to read data from.
p	A pointer to the data buffer.
count	The number of bytes to read from the pipe.
return value	An error code.

DISCUSSION

Note that a pipe need not be connected in order for bytes to be read from it. It is possible for a pipe to have buffered data received from a Newton OS device before the connection was broken. As long as the pipe's state is kCD_Connected or kCD_DisconnectPending, clients of the CDIL are still able to retrieve these bytes.

ERROR CODES

kCD_CDILNotInitialized
kCD_PlatformError
kCD_BadPipeState
kCD_TimeOut
kCD_PipeDisconnected
kDIL_InvalidParameter

kDIL_InvalidHandle
kDIL_OutOfMemory

CD_BytesAvailable

DIL_Error CD_BytesAvailable(CD_Handle *pipe*, long* *count*)

Returns the number of bytes available for reading from the pipe.

pipe	The pipe.
count	A pointer to where the number of bytes available in the pipe should be stored by this function.
return value	An error code.

DISCUSSION

Note that a pipe need not be connected in order for bytes to be read from it. It is possible for a pipe to have buffered data received from a Newton device before the connection was broken. As long as the pipe's state is kCD_Connected or kCD_DisconnectPending, clients of the CDIL are still able to retrieve these bytes.

ERROR CODES

kCD_CDILNotInitialized
kCD_PlatformError
kCD_BadPipeState
kCD_TimeOut
kCD_PipeDisconnected
kDIL_InvalidParameter
kDIL_InvalidHandle
kDIL_OutOfMemory

CD_Write

DIL_Error CD_Write(CD_Handle *pipe*, const void* *p*, long count)

Sends the given bytes to the Newton device.

pipe	The pipe to write data to.
p	A pointer to the data buffer.
count	The number of bytes to write to the pipe.
return value	An error code.

DISCUSSION

The data is not actually sent each time CD_Write is called. It is buffered until either the buffer is full, or a non-CD_Write call is executed: CD_Idle, CD_Read, CD_Disconnect, or CD_BytesAvailable.

ERROR CODES

kCD_CDILNotInitialized
kCD_PlatformError
kCD_BadPipeState
kCD_TimeOut
kCD_PipeDisconnected
kDIL_InvalidParameter
kDIL_InvalidHandle
kDIL_OutOfMemory

CD_FlushOutput

DIL_Error CD_FlushOutput(CD_Handle *pipe*)

Flushes any buffered data written to a pipe.

pipe	The pipe to flush.
return value	An error code.

DISCUSSION

To increase performance, the CDIL buffers all outgoing data. This data remains in the desktop computer until you call CD_FlushOutput to explicitly send to the Newton OS device. Otherwise, the data is implicitly sent on the

next call to CD_Idle, CD_Read, CD_Disconnect, or CD_BytesAvailable. Note that the data could also be sent if the buffer is filled.

ERROR CODES

kCD_CDILNotInitialized
kCD_PlatformError
kCD_BadPipeState
kCD_TimeOut
kCD_PipeDisconnected
kDIL_InvalidParameter
kDIL_InvalidHandle
kDIL_OutOfMemory

CD_Idle

DIL_Error CD_Idle(CD_Handle *pipe*)

Allows the CDIL to service an open connection.

pipe	The pipe to service.
return value	An error code.

DISCUSSION

If the Newton device is sending data very rapidly, you must call this function frequently to buffer that data. The CDIL uses a dynamically sized buffer, but the underlying communication tool may use a statically sized one. If you don't call CD_Idle frequently enough, you may lose data. On the other hand, you unnecessarily slow down your application if you call this function too frequently. Frequencies on the order of a tenth of second should be adequate. In general you calling this function once each time through the main event loop, is sufficient.

ERROR CODES

kCD_CDILNotInitialized
kCD_PlatformError
kCD_BadPipeState
kCD_TimeOut
kCD_PipeDisconnected
kDIL_InvalidParameter
kDIL_InvalidHandle

kDIL_OutOfMemory

CD_GetState

CD_State CD_GetState(CD_Handle *pipe*)

Updates and returns the state of the pipe.

pipe	The pipe whose state you are interested in.
return value	A state constant as listed in “CDIL States” (page 2-9).

DISCUSSION

There is no guarantee that two calls to CD_GetState made one right after the other will return the same value. In particular, the state can always change from kCD_Listening to kCD_ConnectPending or kCD_DisconnectPending, or from kCD_Connected to kCD_DisconnectPending.

CD_GetPlatformError

long CD_GetPlatformError(CD_Handle *pipe*)

Returns the platform-specific error code which caused another CDIL function to return kCD_PlatformError.

pipe	The pipe on which the error occurred. This parameter can be NULL if the error occurred is not associated with a pipe, for example, if a CD_CreateXXX function failed.
return value	A long value containing the platform specific error.

DISCUSSION

The CDIL functions call a wide variety of platform-specific transport functions to implement their functionality. If one of these functions returns an error, the CDIL function returns kCD_PlatformError. You can then, call CD_GetPlatformError to retrieve the actual error code.

ERROR CODES

A platform specific error
kCD_CDILNotInitialized

CD_SetTimeout

DIL_Error CD_SetTimeout(CD_Handle *pipe*, long *timeoutInSecs*)

Sets the timeout period for CD_Read and CD_Write calls in a pipe.

pipe	The pipe whose timeout period is being set.
timeoutInSecs	The timeout period in seconds. The following constants are defined for you: kCD_DefaultTimeout for the default 30 second period, and kCD_NoTimeout if you want calls to CD_Read and CD_Write wait indefinitely.
return value	An error code.

DISCUSSION

When the CDIL pipe is created, it is initialized with a default timeout period of 30 seconds. This timeout period is used to control CD_Read and CD_Write calls (and, indirectly, any flushing of outgoing data). Timeout values are specified on a per-pipe basis.

For CD_Read, if the requested number of bytes are not available after the timeout period, a kCD_Timeout error is returned and no bytes will be transferred. For CD_Write, if no data can be sent after the timeout period, a kCD_Timeout error is returned.

The timeout does not occur, if the data is presently being transferred. That is, a long operation does not fail due to a timeout. Note that an attempt is made to send data even if the timeout is set to zero seconds.

ERROR CODES

kCD_CDILNotInitialized
kCD_PlatformError
kCD_BadPipeState
kCD_TimeOut
kCD_PipeDisconnected
kDIL_InvalidParameter
kDIL_InvalidHandle

CDIL Summary

Type Definitions

CD_Handle

Constants

CDIL States

kCD_Uninitialized
kCD_Disconnected
kCD_Listening
kCD_ConnectPending
kCD_Connected
kCD_DisconnectPending

Timeout Intervals

kCD_DefaultTimeout
kCD_NoTimeout

Error Codes

kDIL_NoError
kDIL_ErrorBase
kDIL_OutOfMemory
kDIL_InvalidParameter
kDIL_InternalError
kDIL_ErrorReadingFromPipe
kDIL_ErrorWritingToPipe
kDIL_InvalidHandle
kCD_ErrorBase
kCD_CDILNotInitialized
kCD_ServiceNotSupported
kCD_BadPipeState
kCD_Timeout

```
kCD_PipeDisconnected
kCD_IndexOutOfRange
kCD_BufferTooSmall
kCD_PlatformError
kCD_TCPCantFindLibraryFns
kCD_TCPInsufficientVersion
kCD_TCPNoSockets
```

Functions

```
DIL_Error CD_Startup()
DIL_Error CD_Shutdown()
DIL_Error CD_CheckADSP()
DIL_Error CD_CheckCTB(const char* toolName)
DIL_Error CD_CheckMNPSerial()
DIL_Error CD_CheckTCP()
DIL_Error CD_GetSerialPortName (long index, char* buffer, long* bufLen)
DIL_Error CD_CreateADSP (CD_Handle* pipe, const char* name,
    const char* type)
DIL_Error CD_CreateCTB (CD_Handle* pipe, const char* toolName,
    const char* configString)
DIL_Error CD_CreateMNPSerial(CD_Handle* pipe, long port, long baud)
DIL_Error CD_CreateTCP(CD_Handle* pipe, long port)
DIL_Error CD_Dispose(CD_Handle pipe)
DIL_Error CD_Disconnect(CD_Handle pipe)
DIL_Error CD_StartListening(CD_Handle pipe)
DIL_Error CD_Accept(CD_Handle pipe)
DIL_Error CD_Read(CD_Handle pipe, void* p, long count)
DIL_Error CD_BytesAvailable(CD_Handle pipe, long* count)
DIL_Error CD_Write(CD_Handle pipe, const void* p, long count)
DIL_Error CD_FlushOutput(CD_Handle pipe)
DIL_Error CD_Idle(CD_Handle pipe)
CD_State CD_GetState(CD_Handle pipe)
long CD_GetPlatformError(CD_Handle pipe)
CD_Error CD_SetTimeout (CD_Handle pipe, long timeoutInSecs)
```