CHAPTER 1

DIL Interface

The Desktop Integration Libraries (DILs) 2.0 are a suite of C-language libraries that desktop applications (for both Windows and Mac OS) can use for data interchange with Newton devices.

About the DILs

There are three libraries that comprise the DILs:

- The CDIL, Communications DIL, allows for a two-way pipe between the desktop application and the Newton device. This pipe imposes no restrictions on the format of the data passed through. Both sides simply read and write any number of bytes. The CDIL can be configured to use a variety of services to implement this pipe: MNP serial, ADSP, TCP, and Communications Toolbox tools.
- The FDIL, Frames DIL, is a desktop implementation of the NewtonScript object model. The FDIL allows you to create and manipulate NewtonScript objects on the desktop machine. This means that the Newton device and desktop applications can send each other NewtonScript objects, and the desktop application can access the data in

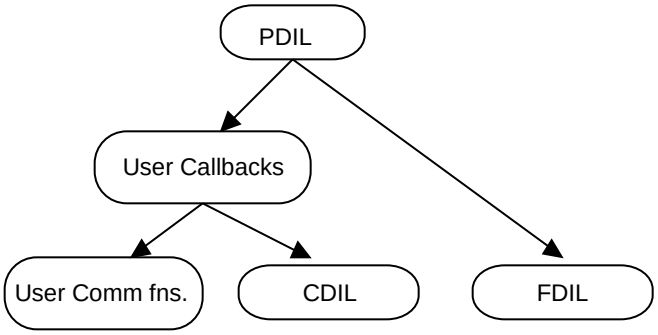
them. The FDIL also provides the ability to stream these objects, allowing you to send the objects through a CDIL pipe, or store them in a file.

- The PDIL, Protocol DIL, is a library that communicates with the Dock application on a Newton device. The Dock application was called Connection in pre-Newton 2.1 OS devices. The PDIL knows about the set of commands (the protocol) used to send information to and receive information from the Dock application. These operations include:
  - connecting to the Dock application
  - getting a list of stores and soups
  - reading the entries stored in the soups
  - performing soup queries and iterating through soup entries with cursors
  - adding new and deleting existing soups and soup entries
  - calling global functions and root view methods
  - downloading packages
  - extending the protocol to execute arbitrary NewtonScript function objects

Interrelationship Between the CDIL, FDIL, and PDIL

The PDIL uses the FDIL directly; the PDIL uses entities such as soup entries and objects retrieved from the Dock application, these are represented with FDIL objects. This is the only strong link between these libraries. The PDIL requires that some sort of link be established with the Newton device, this link can be implemented with the CDIL, but this is not necessary.

Listing 1-1 Dependency between the DIL libraries



Compatibility with 1.x DILs

While the basic responsibilities of the various DILs (that is, CDIL, FDIL, and PDIL) remain the same as their 1.0 counterparts, the actual application program interface (API) have changed significantly. The APIs have been streamlined and their functionality and interfaces have been enhanced.

Information on converting from using the 1.0 to the 2.0 CDIL can be found in “CDIL Compatibility” (page 2-2). The FDIL is sufficiently different from the HLFDIL that no compatibility information can be offered. There was no shipping version of the 1.0 PDIL.

Using the DIL

The DIL is distributed as a single library file since it is considered that these libraries will be used in conjunction. There are a number of versions of this file. There is a normal and debug version for each of the three supported platforms: 68K, PPC, and x86, and dynamic-link versions of the x86 libraries. The header files however, are independent for the different DIL components. The four header files are:

```
DIL.h
CDIL.h
```

FDIL.h
PDIL.h

You only need to include the appropriate header file for the DIL component you are using, CDIL, FDIL, and PDIL; you do not need to explicitly include the DIL.h file itself.

Platform Specific Considerations

On the Mac OS, the libraries are static, built with CodeWarrior Pro 1. The 68K platform, are built using the smart code model, 4-byte ints, 8-byte doubles, 68020 code generations, and SANE floating point numerics. The 68K libraries are in the files:

DIL.68K.Debug
DIL.68K.NoDebug

There are no special considerations on the PowerPC platform, the library files are:

DIL.PPC.Debug
DIL.PPC.NoDebug

The Windows version of the library comes in static and dynamic link versions. They were built with Visual C++ 5.0, and should work in any compatible development environment. The static libraries have been compiled against the multi-threaded static C Runtime libraries: “Multi-threaded Debug” for the debug version, and “Multi-threaded” for the non-debug version. If you use these static-link libraries, you'll need to use matching settings in your “C/C++” / “Code Generation” settings panel. The static library files are:

DIL2.lib
DIL2D.lib

There are no special considerations with the dynamic link libraries. The dynamic library files are:

DIL2.dll
DIL2.lib
DIL2D.dll
DIL2D.lib

Note
The dynamic link version of the "DIL2.lib" and "DIL2D.lib" contain stub functions, and are thus different from the files with the same name in the static libraries directory. ◆

If you use the Windows static link library you must defined the macro `USING_STATIC_DIL` before including any of the DIL header files. If you are using the Windows dynamic link libraries, you must similarly define the macro `USING_DYNAMIC_DIL` before including any DIL header files. Failure to define either of these symbols results in an error message reminding you to define one or the other.

Debug Versions of the Library

The DIL includes a debug version. To use the debug version of the library, include the proper library file, and define the macro `DIL_ForDebug` before including any of the DIL header files. The debug versions are:

```
DIL.68K.Debug
DIL.PPC.Debug
DIL2D.dll
DIL2D.lib
```

The debug versions of the library includes embedded asserts. If something terribly wrong occurs, the DIL calls the Standard C Library `assert` facility. There are also a number of changes in the debug version of the FDIL component; for information on these changes, see "The Debug Version of the FDIL" (page 3-26). You should only ship code built with the non-debug version of the library.