DIL 2.0 Release Notes

February 13, 1998

CDIL

- See the note at the end of this document describing the recommended method of disconnecting from the Newton device.

FDIL

- There is a subtle difference between creating slot names using FD_MakeSymbol() and NewtonScript syntax on a Newton device. FD_MakeSymbol will make valid symbols using the string passed in (e.g. FD_MakeSymbol("1234") where NewtonScript requires vertical bars to make the same symbol (sym := '|1234|).

  On the desktop, the vertical bar characters are not only not required, but if used will become part of the symbol. FD_MakeSymbol("|1234|") does not equal '|1234| on the Newton device.

PDIL

- Registering a duplicate protocol extension (i.e. with the same extension ID as one already registered) will cause the connection to drop.

- PD_Idle is a multi–purpose function, returning status, unexpected commands, and error codes. To be complete, you should look at the result of PD_Idle as follows:
  - < 0 is an error
  - = 0 is okay (i.e. nothing from the Newton device)
  - = 1 is an AutoDock command from the Newton device
  - = 2 is a Cancel from the Newton device
  - = 3 is a Disconnect from the Newton device
  - = 4 is a Hello command from the Newton device
  - > 4 is an unexpected command, typically sent from a protocol extension

- Hello is a command sent by the Newton device when it is idling, unless it has been told that the desktop is in control. The PDIL always puts the desktop in control, so you should never receive one of the Hello commands.

- When adding a frame containing a large binary object (using the PD_AddEntry call) the large binary must have a class name. PD_AddEntry will fail if a class name is not specified:

```
bin1 = FD_MakeLargeBinary(BIN1SIZE, "text", kFD_NoCompression);
entry = FD_MakeFrame();
FD_SetFrameSlot(entry, "b", bin1));
err = PD_AddEntry(gSession, entry, &id);
```


- When PD_DeleteEntries is called with an array that includes _uniqueIDs that don't match what's in the soup, it will go ahead and delete the next soup entry in the cursor list — one that was not at all in the list of entries to be deleted. This is because the DOCK application simply does a Cursor GotoKey on the specified ID, which returns the specified entry, or the next higher entry.

- Protocol extensions must return NIL. Failure to do so can cause erratic behavior on the Newton device. Here is an example of a very simple protocol extension:

```
output := func(ep)
begin
   local nBeeps := ep:ReadCommandData();
   for i := 1 to nBeeps do
      GetRoot():SysBeep();
   ep:WriteCommand("BEEP", nBeeps, true);
   return nil;
end;
```

  From your program, you would make this call:

```
FD_Handle result;
PD_CallExtension(gSession, 'BEEP', FD_MakeInt(10), &result)
```

  Upon return, result will contain the integer 10.

- Your program should follow a well–behaved method of disconnecting from the Newton device. Failure to do so can cause the Newton device to restart, especially when connected to Windows. This was the source of all of the outstanding bugs in the 2.0a7 release notes. This is the recommended method of disconnecting:

```
// eat any commands that might be backed up in the pipe
do {
      status = PD_Idle(gSession);
      // you probably want to put a time-out mechanism
      // here to avoid an infinite loop
} while (status != kPD_Okay);

// send the disconnect command and wait for the connection to drop
err = PD_Dispose(gSession);
do {
      err = CD_Idle(gPipe);
      state = CD_GetState(gPipe);
while (state == kCD_Connected);

// finally, close the connection
err = CD_Disconnect(gPipe);
```

