Newton 1.0 Connection Protocol

The Connection protocol is used to communicate between the desktop and Newton.

This document should be read in conjunction with DockProtocol.h which defines the constants and structures referenced here.

NOTE This protocol has been superseded by the 2.0 Newton ROM: refer to the Dante Connection Protocol document.

Protocol Overview

Newton communicates with the desktop by exchanging Newton event commands. The general command structure looks like this:

```
ULong     'newt'        // event header
ULong     'dock'        // event header
ULong     'aaaa'        // specific command
ULong      length       // the length in bytes of the following data
UChar      data[]       // data, if any
```

NOTE

- The length associated with each command is the actual length in bytes of the data following the length field.
- Data is padded with nulls to a 4 byte boundary.
- Multi-byte values are in big-endian order.
- Strings are null-terminated 2-byte UniChar strings unless otherwise specified.
- NewtonScript objects are sent in Newton Streamed Object Format (NSOF) (see the Newton Formats document, chapter 4).

Desktop Applications

Several desktop applications that provide connection services to Newton are available, some of them in Apple's archive. They all implement the protocol defined in this document.

Newton Connection Kit (NCK)	1.0	

Protocol: 1

Functions: backup, restore, install

Newton Package Installer (NPI)	1.1	released June 20, 1994

Protocol: 1

Functions: install package only

Newton Connection for Mac OS X (NCX)	2.0.2	released August 8, 2013

Protocol: 1 & 2 link
Functions: backup, restore, install, import, export, keyboard passthrough, screenshot

---

Connection Protocol

A Newton docking session performs one operation and then disconnects.

Every session starts like this:

```
       Desktop                    Newton
                            <  kDRequestToDock
    kDInitiateDocking       >
                            <  kDNewtonName
```

At this point the desktop can specify a timeout — the time after which if there are no events the connection should be deemed to be broken:

```
    kDSetTimeout            >
                            <  kDResult
```

or if no timeout is required, the desktop can simply send a `kDResult`.

```
    kDResult                >
```

A typical synchronize session might continue like this:

```
       Desktop                    Newton
    kDGetStoreNames         >
                            <  kDStoreNames
    kDLastSyncTime          >    this one's fake (0) just to get the Newton time
                            <  kDCurrentTime
    kDSetCurrentStore       >
                            <  kDResult
    kDLastSyncTIme          >
                            <  kDCurrentTime
    kDGetPatches            >
                            <  kDPatches
    kDGetPackageIDs         >
                            <  kDPackageIDList
    kDBackupPackages        >
                            <  kDPackage
    kDBackupPackages        >
                            <  kDPackage
    kDBackupPackages        >
                            <  kDResult
    kDGetSoupNames          >
                            <  kDSoupNames
    kDGetInheritance        >
                            <  kDInheritance
    kDSetCurrentSoup        >
                            <  kDResult
    kDGetSoupInfo           >
                            <  kDSoupInfo
    kDGetSoupIDs            >
                            <  kDSoupIDs
    kDGetChangedIDs         >
                            <  kDChangedIDs
    kDDeleteEntries         >
                            <  kDResult
```

```
kDAddEntry             >
                       <  kDAddedID
kDReturnEntry          >
                       <  kDEntry
kDDisconnect           >
```

A restore session would look like this:

```
   Desktop                 Newton
kDGetStoreNames        >
                       <  kDStoreNames
kDSetCurrentStore      >
                       <  kDResult
kDDeleteAllPackages    >
                       <  kDResult
kDGetSoupNames         >
                       <  kDSoupNames
kDSetCurrentSoup       >
                       <  kDResult
kDEmptySoup            >
                       <  kDResult
kDAddEntry             >
                       <  kDResult
kDDeletePkgDir         >
                       <  kDResult
kDLoadPackage          >
                       <  kDResult
kDDisconnect           >
```

A load package session would look like this:

```
   Desktop                 Newton
kDLoadPackage          >
                       <  kDResult
kDDisconnect           >
```

Command Summary

The following is a summary of all the commands that can be used and their four-letter definitions:

Newton > Desktop

```
kDRequestToDock        'rtdk'
kDNewtonName           'name'     // + name of the Newton
kDCurrentTime          'time'     // + current time on the Newton
kDInheritance          'dinh'     // + array of class, superclass pairs
kDPatches              'patc'     // + patch package

kDStoreNames           'stor'     // + array of store names & signatures

kDSoupNames            'soup'     // + array of soup names & signatures
kDIndexDescription     'indx'     // + index description array

kDSoupIDs              'sids'     // + array of ids for the soup
kDChangedIDs           'cids'     // + array of ids
kDResult               'dres'     // + error code
kDAddedID              'adid'     // + the id of the added entry
kDEntry                'entr'     // + entry being returned

kDPackageIDList        'pids'     // + list of package ids
kDPackage              'apkg'     // + package
```

Desktop > Newton

```
kDInitiateDocking       'dock'      // + session type
kDSetTimeout            'stim'      // + timeout in seconds
kDLastSyncTime          'stme'      // + time of last sync
kDGetInheritance        'ginh'
kDGetPatches            'gpat'

kDGetStoreNames         'gsto'
kDSetCurrentStore       'ssto'      // + store frame

kDGetSoupNames          'gets'
kDSetCurrentSoup        'ssou'      // + soup name
kDCreateSoup            'csop'      // + name + index description
kDEmptySoup             'esou'
kDDeleteSoup            'dsou'

kDGetSoupInfo           'gsin'
kDGetIndexDescription   'gind'
kDGetSoupIDs            'gids'
kdGetChangedIDs         'gcid'
kDDeleteEntries         'dele'      // + list of IDs
kDAddEntry              'adde'      // + soup entry
kDReturnEntry           'rete'      // + ID to return
kDReturnChangedEntry    'rcen'      // + ID to return

kDLoadPackage           'lpkg'      // + package
kDGetPackageIDs         'gpid'
kDBackupPackages        'bpkg'
kDDeleteAllPackages     'dpkg'
kDDeletePkgDir          'dpkd'

kDDisconnect            'disc'
```

Desktop < > Newton

```
kDSoupInfo              'sinf'      // + soup info frame
kDChangedEntry          'cent'      // + soup entry

kDResult                ‘dres’      // + error code
kDHello                 'helo'
kDTest                  'test'      // + variable length data
```

Dock Commands

All commands begin with the `'newt'`, `'dock'` event header as shown in the general form. For simplicity, that's not shown in the descriptions that follow.

Session Initiation

kDRequestToDock

Desktop	<	Newton
		ULong 'rtdk'
		ULong length = 4
		ULong protocol version

The Newton initiates a session by sending this command to the desktop, which is listening on the network, serial, etc. The protocol version is the version of the messaging protocol that's being used by the Newton ROM. The desktop sends a `kDInitiateDocking` command in response.

kDInitiateDocking

Desktop **>** **Newton**

```
ULong    'dock'
ULong    length = 4
ULong    session type
```

The session type can be one of {none, settingUp, synchronize, restore, loadPackage, testComm, loadPatch, updatingStores}; see the Session type enum in DockProtocol.h The Newton responds with information about itself.

kDNewtonName

Desktop **<** **Newton**

```
ULong    'name'
ULong    length
struct   NewtonInfo
UniChar  name[]
```

The Newton's name can be used to locate the proper synchronize file. The version info includes things like machine type (e.g. J1), ROM version, etc; see the NewtonInfo struct in DockProtocol.h

kDSetTimeout

Desktop **>** **Newton**

```
ULong    'stim'
ULong    length = 4
ULong    timeout in seconds
```

This command sets the timeout for the connection (the time the Newton will wait to receive data before it disconnects). This time is usually set to 30 seconds.

System State Operations

kDGetPatches

Desktop **>** **Newton**

```
ULong    'gpat'
ULong    length = 0
```

This command requests the system patches.

kDPatches

Desktop **<** **Newton**

```
ULong    'patc'
ULong    length
?
```

Undocumented.

kDGetInheritance

Desktop **>** **Newton**

```
ULong    'ginh'
ULong    length = 0
```

This command requests the inheritance frame.

kDInheritance

Desktop	<	Newton
		ULong 'dinh' ULong length ?

Undocumented.

Store Operations

kDGetStoreNames

Desktop	>	Newton
ULong 'gsto' ULong length = 0		

This command requests information (not just names!) about all the stores on the Newton.

kDStoreNames

Desktop	<	Newton
		ULong 'stor' ULong length NSOF array of frames

This command is sent in response to a `kDGetStoreNames` command. It returns information about all the stores on the Newton. Each array slot contains the following information about a store:

```
{ name: "",
  signature: 1234,
  totalSize: 1234,
  usedSize: 1234,
  kind: "",
  info: {store-info-frame},
  readOnly: true,
  defaultStore: true,         // only for the default store
  storePassword: password     // only if a store password has been set
}
```

kDLastSyncTime

Desktop	>	Newton
ULong 'gsto' ULong length = 0		

This command requests the time the current store was last backed up.

kDCurrentTime

Desktop	<	Newton
		ULong 'time' ULong length = 4 ULong time in minutes since 1 Jan 1904

kDSetCurrentStore

```
        Desktop         >       Newton
        ULong    'ssto'
        ULong    length
        NSOF     store frame
```

This command sets the current store on the Newton. A store frame is sent to uniquely identify the store to be set:

```
{ name: "foo",
  kind: "bar",
  signature: 1234,
  info: {store-info-frame}    // this one is optional
}
```

kDGetSoupNames

```
        Desktop         >       Newton
        ULong    'gets'
        ULong    length = 0
```

This command is sent when a list of soup names is needed. It expects to receive a `kDSoupNames` command in response.

kDSoupNames

```
        Desktop         <       Newton
                                ULong    'soup'
                                ULong    length
                                NSOF     array of name strings
                                NSOF     array of soup signature integers
```

This command is sent in response to a `kDGetSoupNames` command. It returns the names and signatures of all the soups in the current store.

Soup Operations

kDCreateSoup

```
        Desktop         >       Newton
        ULong    'csop'
        ULong    length of name
        UniChar  name[]  // aligned on 4-byte boundary
        NSOF     soup indexes
```

This command is used to create a new soup. The soup name should be padded to an even multiple of 4 by adding zero bytes to the end of the name string.

kDEmptySoup

```
        Desktop         >       Newton
        ULong    'esou'
        ULong    length
        UniChar  name[]
```

This command is used by restore to remove all entries from a soup before the soup data is restored.

kDDeleteSoup

Desktop **>** **Newton**

```
ULong    'dsou'
ULong    length
UniChar  name[]
```

This command is used by restore to delete a soup if it exists on the Newton.

kDSetCurrentSoup

Desktop **>** **Newton**

```
ULong    'ssou'
ULong    length
UniChar  name[]
```

This command sets the current soup. Most of the other commands pertain to this soup so this command must preceed any command that uses the current soup. If the soup doesn't exist a `kDSoupNotFound` error is returned but the connection is left alive so the desktop can create the soup if necessary. Soup names must be < 25 characters.

kDGetSoupInfo

Desktop **>** **Newton**

```
ULong    'gsin'
ULong    length = 0
```

This command requests info for the current soup..

kDSoupInfo

Desktop **<** **Newton**

```
ULong    'sinf'
ULong    length
NSOF     soup info frame
```

This command is used to return a soup info frame from the Newton. When received the info for the current soup is set to the specified frame.

kDSetSoupGetInfo

Desktop **>** **Newton**

```
ULong    'ssgi'
ULong    length
UniChar  name[]
```

This command is like a combination of `kDSetCurrentSoup` and `kDGetChangedInfo`. It sets the current soup--see `kDSetCurrentSoup` for details. A `kDSoupInfo` or `kDResult` command is sent by the Newton in response.

kDGetChangedInfo

Desktop **>** **Newton**

```
ULong    'cinf'
ULong    length = 0
```

This command is like `kDGetSoupInfo` except that it only returns the soup info if it has been changed since the time set by the `kDLastSyncTime` command. If the info hasn't changed a `kDResult` with 0 is returned.

kDGetIndexDescription

```
Desktop          >          Newton
ULong    'gidx'
ULong    length = 0
```

This command requests the definition of the indexes that should be created for the current soup.

kDIndexDescription

```
Desktop          <          Newton
                            ULong    'didx'
                            ULong    length
                            NSOF     indexes
```

This command specifies the indexes that should be created for the current soup.

kDGetChangedIndex

```
Desktop          >          Newton
ULong    'cidx'
ULong    length = 0
```

This command is like `kDGetIndexDescription` except that it only returns the index description if it has been changed since the time set by the `kDLastSyncTime` command. If the index hasn't changed a `kDResult` with 0 is returned.

Entry Operations

kDGetSoupIDs

```
Desktop          >          Newton
ULong    'gids'
ULong    length = 0
```

This command is sent to request a list of entry IDs for the current soup. It expects to receive a `kDSoupIDs` command in response.

kDSoupIDs

```
Desktop          <          Newton
                            ULong    'sids'
                            ULong    length
                            ULong    count of elements in the ids array
                            ULong    ids[]
```

This command is sent in response to a `kDGetSoupIDs` command. It returns all the entry IDs from the current soup.

kDGetChangedIDs

```
Desktop          >          Newton
ULong    'gcid'
ULong    length = 0
```

This command is sent to request a list of changed IDs for the current soup. It expects to receive a `kDChangedIDs` command in response.

kDChangedIDs

Desktop **<** **Newton**

```
                             ULong    'cids'
                             ULong    length
                             ULong    count of elements in the ids array
                             ULong    ids[]
```

This command is sent in response to a `kDGetChangedIDs` command. It returns all the ids with mod time > the last sync time. If the last sync time is 0, no changed entries are returned (this would happen on the first sync).

kDDeleteEntries

Desktop **>** **Newton**

```
ULong    'dele'
ULong    length
ULong    count of elements in the ids array
ULong    ids[]
```

This command is sent to delete one or more entries from the current soup.

kDAddEntry

Desktop **>** **Newton**

```
ULong    'adde'
ULong    length
NSOF     entry
```

This command is sent when the PC wants to add an entry to the current soup.

kDAddedID

Desktop **<** **Newton**

```
                             ULong    'adid'
                             ULong    length = 4
                             ULong    id
```

This command is sent in response to a `kDAddEntry` command from the PC. It returns the ID that the entry was given when it was added to the current soup.

kDReturnEntry

Desktop **>** **Newton**

```
ULong    'rete'
ULong    length = 4
ULong    id
```

This command is sent when the PC wants to retrieve an entry from the current soup.

kDEntry

Desktop **<** **Newton**

```
                             ULong    'entr'
                             ULong    length
                             NSOF     entry
```

This command is sent in response to a `kDReturnEntry` command. The entry in the current soup specified by the ID in the `kDReturnEntry` command is returned.

kDReturnChangedEntry

Desktop **>** **Newton**

```
ULong    'rcen'
ULong    length = 4
ULong    id
```

This command is sent when the PC wants to retrieve a changed entry from the current soup.

kDChangedEntry

Desktop **< >** **Newton**

```
ULong    'cent'
ULong    length
NSOF     entry
```

This command is sent by the Newton in response to a `kDReturnChangedEntry` command from the desktop. It can also be sent by the desktop.

Package Operations

kDGetPackageIDs

Desktop **>** **Newton**

```
ULong    'gpid'
ULong    length = 0
```

This command is sent to request a list of package ids. This list is used to remove any packages from the PC that have been deleted on the Newton.

kDPackageIDList

Desktop **<** **Newton**

```
ULong    'pids'
ULong    length
ULong    count
NSOF     package id frames
```

This command sends a list of package frames to the desktop. For each package the information sent is this:

```
ULong    packageSize;
ULong    packageId;
ULong    packageVersion;
ULong    format;
ULong    deviceKind;
ULong    deviceNumber;
ULong    deviceId;
ULong    modifyDate;
ULong    isCopyProtected;
ULong    length;       // length in bytes of name
UniChar  name[];
```

Note that this is not sent as an array! It's sent as binary data. Note that this finds packages only in the current store.

SIMON'S NOTE It is unclear from this description exactly what form the id list takes, and I have not used this command so cannot comment.

kDBackupPackages

Desktop **>** **Newton**

```
ULong    'bpkg'
ULong    length = 0
```

This command is sent to backup any packages that are installed on the Newton. It expects a `kDPackage` command or a `kDResult` with an error of 0 (to indicate that there are no more packages) in response.

kDPackage

Desktop **<** **Newton**

```
ULong    'apkg'
ULong    length
ULong    package ID
ULong    length in bytes of following name
UniChar  name[]
NSOF     package frame
```

This command sends a package to the desktop. It's issued repeatedly in response to a `kDBackupPackages` command.

kDLoadPackage

Desktop **>** **Newton**

```
ULong    'lpkg'
ULong    length
UChar    package data []
```

This command will load a package into the Newton's RAM. The package data should be padded to an even multiple of 4 by adding zero bytes to the end of the package data.

kDDeleteAllPackages

Desktop **>** **Newton**

```
ULong    'dpkg'
ULong    length = 0
```

This command is used by restore to delete all installed packages from the Newton. It expects a `kDResult` with an error code in response.

kDDeletePkgDir

Desktop **>** **Newton**

```
ULong    'dpkd'
ULong    length = 0
```

This command is used by restore to delete the directory of installed packages from the Newton. It expects a `kDResult` with an error code in response.

General Operations

kDResult

Desktop **< >** **Newton**

```
ULong    'dres'
ULong    length = 4
SLong    error code
```

This command is sent by either Newton or PC in response to any of the commands that don't request data. It lets the requester know that things are still proceeding OK.

kDHello

Desktop **< >** **Newton**

```
ULong    'helo'
ULong    length = 0
```

This command is sent during long operations to let the Newton or desktop know that the connection hasn't been dropped.

kDTest

Desktop **< >** **Newton**

```
ULong    'test'
ULong    length
NSOF     object
```

This command is first sent from the desktop to the Newton. The Newton immediately echos the object back to the desktop. The object can be any NewtonScript object (anything that can be sent through object read/write).

This command can also be sent with no ref attached. If the length is 0 the command is echoed back to the desktop with no ref included.

Session Termination

kDDisconnect

Desktop **>** **Newton**

```
ULong    'disc'
ULong    length = 0
```

This command is sent to the Newton when the docking operation is complete.