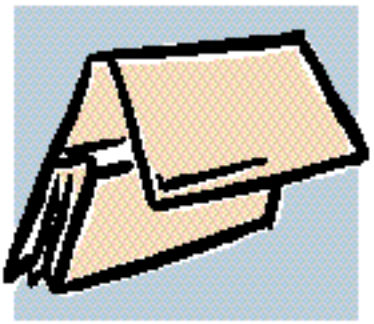
Accordian



A live link between your Newton PDA and FileMaker Pro Databases

Version 1.0 User Guide

NMI & Revelar, Inc. License Agreement

Before using this software, please read the License Agreement (“Agreement”) in the Read Me First! text file in the Accordian 1.0 Folder you now have installed on your computer. Your use of the software implies that you agree to be bound by the Agreement. If you do not accept the Agreement, promptly return the product for a full refund (less shipping).

Accordian and RCU were written by Van C. Evans and W. Bryant Eastham. Documentation was written by Sally Miller and Van C. Evans.

Thanks also to Andrea Evans, Annette Eastham, Robyn Buckwalter, Joe Patterson, and Ed Firmage of NMI.

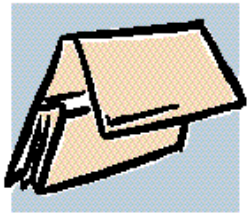
Technical Support

Please read carefully the instructions in this guide. It was written to be straightforward and intuitive. If you experience any problems in the installation process or the normal use of the software that we have not explained, feel free to call. We are committed to supply telephone support free of charge, *provided* you return your completed registration card. You may call Monday through Friday from 9:00 a.m. to 5:00 p.m. Mountain Time, at these numbers:

800-669-5191
801-278-7102

Have your serial number handy, and where possible, be seated at your computer with the Newton PDA ready. We have found that we can help you best when we walk through the problem right on your screen. It is not uncommon for us to take your name and number and then return your call, when someone is available to help you. If this is the case, please be patient with us, since we will return your call as soon as possible. You may also submit an email at:

www.revelar.com/support.html



Welcome

Thank you for choosing Accordian for the Macintosh® and Newton systems. Accordian gives you a real time connection to the applications of your Newton PDA from your Macintosh FileMaker Pro databases.

Features For Version 1.0

- Transfer data from your Newton to FileMaker Pro 3.0 databases without creating exported text files.
- View overviews of your Newton data. View by folder, range, and use the Find command to search for any text in the selected application.
- Add menu items in FileMaker Pro to control Newton data.
- Synchronize the Date and Time of your Newton with your Macintosh.
- Third-party software support. Developers: write an "ADF" file in just a few hours and let Accordian be your connection to the desktop.
- Folder support. Include the Folder name, where applicable, in your imports and exports.
- Transfer Note fields while retaining return and tab information.

System Requirements

- Macintosh with 68040 or Power PC processor.
- 16 Mb RAM
- System 7.1 or greater
- FileMaker Pro 3.0; Version 4.0 not supported
- Apple Modem Tool (place this in your extensions folder)
- Apple MessagePad 120 (2.0), MP130, MP2000, MP2100

Installation

Accordian for your Macintosh uses the RCU 2.3 Newton software to connect to. You must install this onto your Newton. Users of the Revelar Connection Utility need not download it again. You can install the RCU2.3.pkg on your Newton PDA by using one of the following:

- Newton Backup Utility (NBU)
- Newton Connection Utility (NCU)
- Newton Package Installer (NPI)
- RCU 2.x or greater

If you choose RCU to install the package, you must connect to the "Connection" or "Dock" application on the Newton. After that, you can connect to RCU on the Newton to install other packages. You must have a standard Macintosh serial cable. The cable that came with your MessagePad will do.

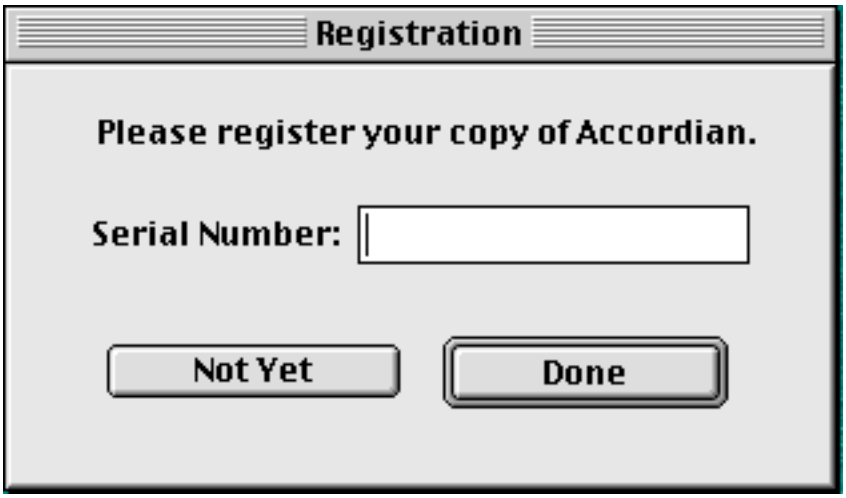
If you intend to have Accordian connected for an extended period of time, we recommend you plug your Newton device into an AC power adapter. If you do not, keep in mind that your Newton PDA will shut off at the sleep interval you set in preferences if there is no activity in Accordian for that interval. Simply being connected does not keep your Newton awake.

Registration

To register your copy of Accordian:

1. Launch Accordian on your Macintosh.

The Registration window opens as shown here:



2. Type in your serial number. Please enter all letters in upper case, and there are no letter "O"s, only zeros.

The serial number is a license to use Accordian on a single Newton PDA. If you want Accordian for use on another Newton PDA, you must buy another copy. Enter your serial number in upper case letters.

3. Press the Done button to close the Registration window.

If you press the "Not Yet" button, you will not be able to transfer records to and from the Newton to FileMaker Pro. The program will, however, be fully functional up to that point.

Connecting to the Macintosh

This section assumes you have read and understand the Newton's basic operations and have already downloaded the RCU2.3.pkg file into your Newton device and registered the software. If you have not already done this, refer to "Installation" and "Registration" in this guide.

1. Plug one end of your serial cable into the modem or printer port of your Macintosh and the other end into your Newton.

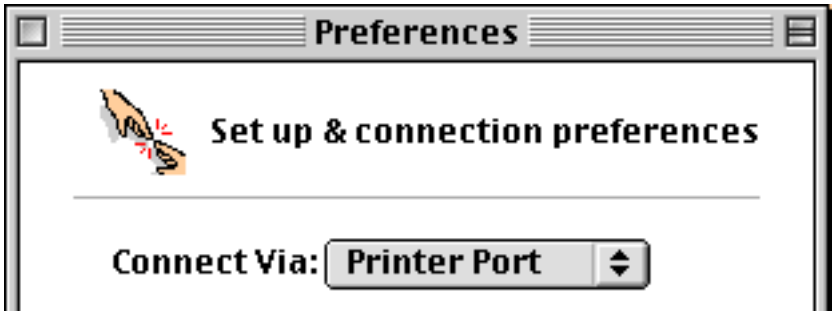
The cable end is cylindrical with one flat side, or has arrows pointing up. Place the flat side up to plug in the cable. You should have already accomplished this step to download the RCU2.3.pkg. We recommend you use the modem port, since you may wish to print reports while using Accordian.

You must have the Apple Modem Tool installed in your System Folder (Extensions folder) and you should turn AppleTalk off of the port you are trying to connect to. The Apple Modem Tool is found on the NBU or NCU disk that came with your Newton PDA. Turn AppleTalk off through the Chooser.

2. Double-click the Accordian icon from the Finder to launch the program on your Macintosh, if it is not already launched.

You must have at least 4 Mb RAM available on your Macintosh to properly use Accordian. RAM requirements will be more if you use high-resolution or a high number of colors on your monitor.

3. Select Preferences... under the File menu to open the Preferences window.



4. Select the desired serial port on the Connect via: pop-up menu.

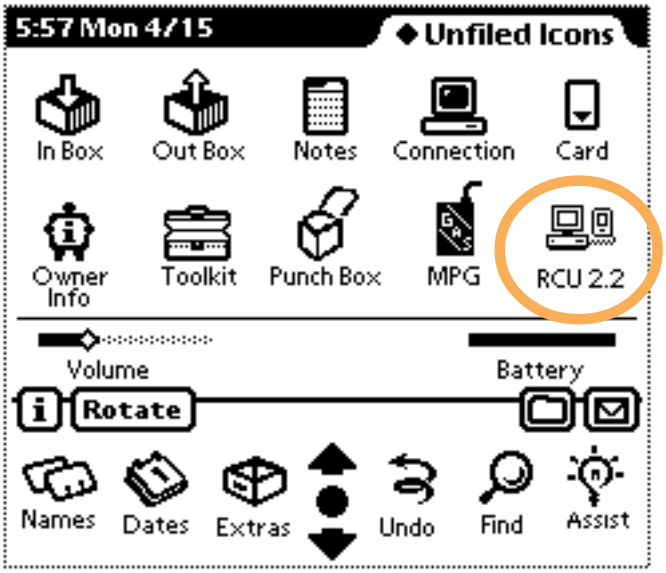
Make certain the port you select is the same port you have plugged your serial cable into. The next time you launch your Accordian program, the Connect via: menu item will default to the last selection you made.

From the Preferences window, you may also check on the Auto-connect at start up button. With this preference, Accordian automatically attempts connection to your Newton as soon as you launch the application. You will still need to press the connect button on the Newton RCU side.

5. Close the Preferences window.

6. Turn on your Newton, if it is not already on, and tap the Extras button to open the Extras drawer, if it is not already open.

7. Tap the RCU 2.3 icon to launch the program on your Newton device.



Trouble Shooting: If you do not see the RCU 2.3 icon, try reinstalling the application. If you installed RCU 2.3 on a PC card, make sure that the card has not been removed.

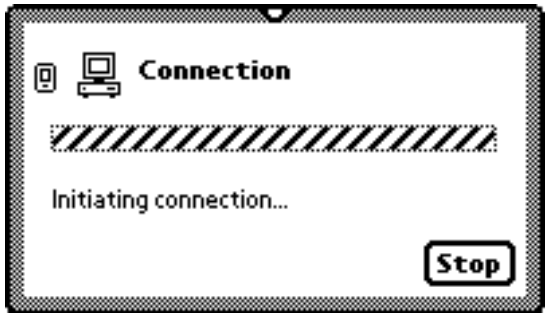
Once you launch RCU 2.3 on your Newton, the following window displays:

8. Press the Connect button in the Accordian window.

The status text on your Macintosh changes to: Waiting for connection….

9. Tap Connect on your Newton to start the connection.

A window on your Newton shows the connection progress.



Once the connection is made, the status text on your Macintosh changes to read: You are connected to RCU on the Newton. The status text and a window on your Newton informs you if a connection isn't made.

Note: You will note that the Newton performs sluggishly while connection is made.

Trouble Shooting: If a connection is not made, check that the cables are properly attached and make sure no other programs are tying up the port. AppleTalk must be inactive for the port you are trying to connect to (AppleTalk is inactivated through Chooser), and the Apple Modem Tool should be installed in your System Folder (Extensions folder). If you have a Duo or PowerBook with an internal modem, you may have to turn off the modem first.

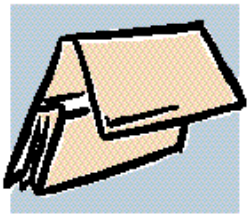
Disconnecting

To properly disconnect the link between your Macintosh and Newton, either select the Disconnect button in the Accordian window or press the Disconnect button on your Newton. You will note that the Newton performs sluggishly while connection is made.

You need not select Disconnect on both your Macintosh and Newton. Disconnecting from one or the other sends a signal for both to disconnect. Unlinking may take a few seconds.

Connection With Batteries

If your Newton is running on batteries and is set to "sleep" within a designated time (i.e., after 1, 5, or 10 minutes of inactivity), a connection to Accordian won't necessarily keep your Newton on if you are not actively using Accordian. Unlike your Macintosh, which stays on until you turn it off, the sleep function turns off your Newton when it is not active in order to conserve battery power. While you use Accordian on your Macintosh, such as importing entries to your Newton, the sleep function on your Newton will not turn on. The sleep function also won't go into effect if your Newton is plugged into an A/C adapter.

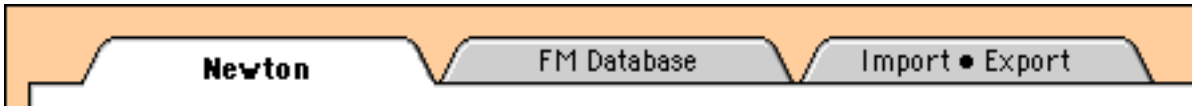


Basic Use

This section explains the components of the Accordian window, including the buttons, status text, tables, and pop-up menus.

General Design and Layout

The main window has three tabs. It is like an accordion* file folder with different compartments—each separated by a tab. Think of it as having data available in each compartment to transfer to another compartment. By default, the Newton tab is shown and its name is in boldface to indicate Newton is the front-most tab. Click a tab to switch to a different tab in the window.



Each of the three tabs has a distinct purpose. Step through them from left to right to accomplish your task of transferring data between the Newton and the desktop. Each tab's purpose is briefly given here:

- Newton. To begin with, make a connection to your Newton and select the Newton application you want to work with. Then narrow down the data you wish to export to FileMaker by restricting all the records in the application using folders, ranges, and performing a search.

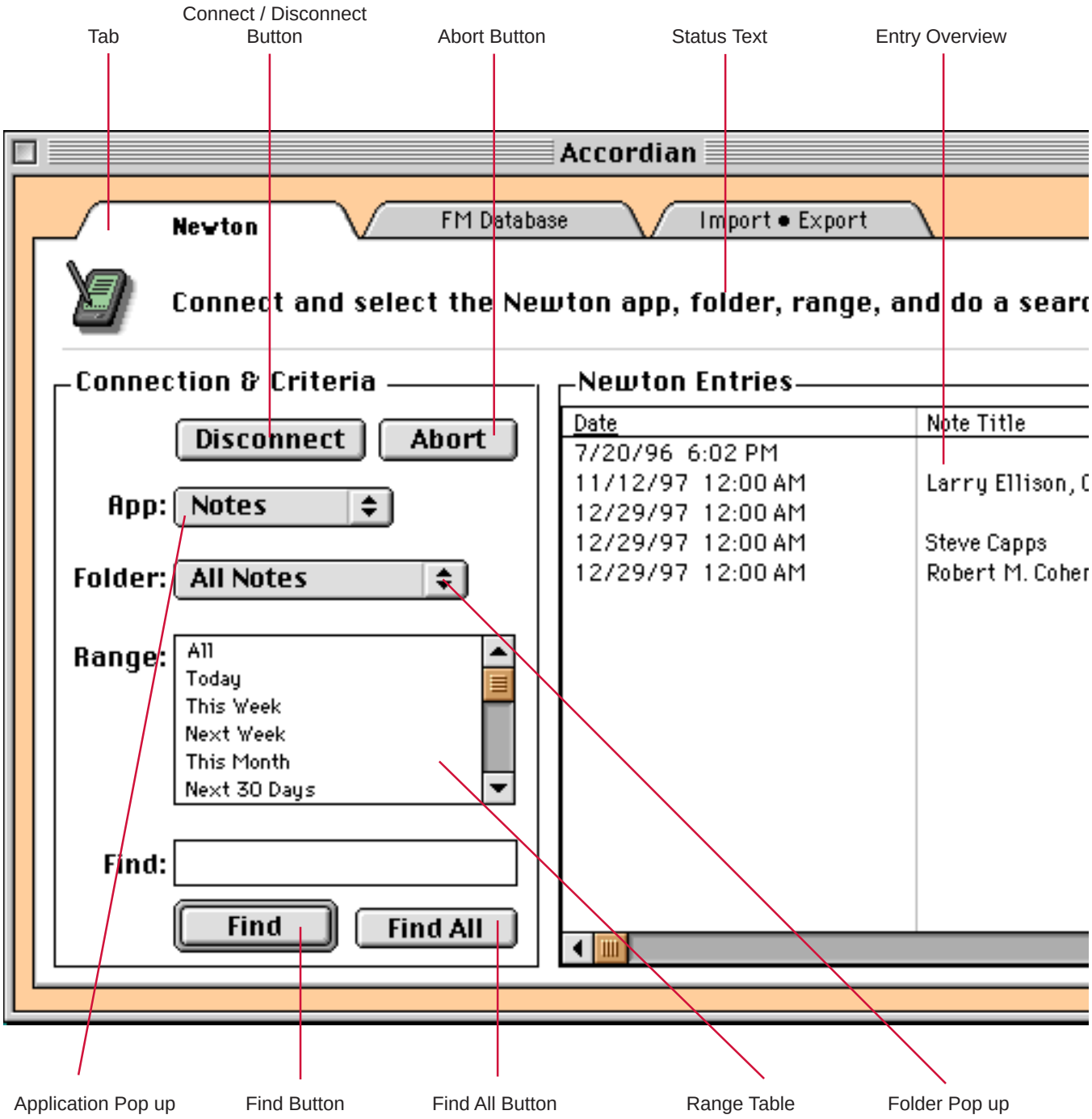
- FM Database. With FileMaker running in the background, retrieve and select from a list of open databases. Then select a layout from that database. Fields are then listed by type to help you in your field matching decisions.

- Import • Export. After performing the steps in the first two tabs, you are ready to match or map together fields from the Newton to fields in the FileMaker database you have chosen. Once accomplished, you press a button to transfer the data to the Newton or the desktop.

* Yes, we know it is spelled "Accordion." Besides avoiding potential name conflicts, we didn't want our software confused with polka music. Hence, the "a."

Newton Tab

Once you have linked up, Accordian gives you access to the Newton's built in applications of Names, Notes, Calendar, etc. The main window of Accordian, which appears on your Macintosh, displays the information that is stored on your Newton. This is a "virtual" window, which means that no information is stored on your Macintosh unless you export the data into FileMaker.



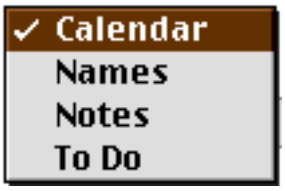
Making a Connection & Specifying Newton Entries

Since Accordian is an application designed specifically for Newton devices, tasks are decidedly Newton-centric; that is, the Newton is the center of the Accordian world. You import to the Newton or export from the Newton to FileMaker. It may take some getting used to, but for our purposes here, we will always refer to importing and exporting that way. No matter which way you want to transfer records, you will need to make a connection to the Newton. If you are importing, after making a connection, you only need to select the Newton application in this tab (step 2 below). All else is handled by the matching of fields in the Import•Export tab. If you are exporting, you will need to specify which Newton entries you will export to FileMaker as explained here:

1. Make a connection to your Newton as explained on pages 5-7.

2. Select from the App: pop up which Newton application you want to work with.

Accordian remembers the last application used before disconnecting. The first time, however, the Calendar application will display since applications are listed in alphabetical order.



3. Select a folder item under the Folder: pop up menu.

This menu displays the folders in your Newton that apply to a particular application. Selecting a menu item retrieves everything in that folder and loads it into the Entry Overview.

Accordian always defaults to the Unfiled folder (e.g., Unfiled Names). Selecting the All folder (e.g., All Names) displays all entries in the overview. Ranges and Find work only within the entries in the selected folder (i.e., within the current overview).

The Folders menu disables for applications that don't store information in folders, such as the Calendar application, which doesn't store appointments in folders.

4. Select a desired range.

When you specify a range, the overview displays those entries that fall within the range. If an application doesn't support ranges, the range table appears with no options. Accordian always defaults to the first range in the list, although it is not selected. You cannot have an "All" range in Calendar or Dates, because repeating appointments will calculate and display infinitely.

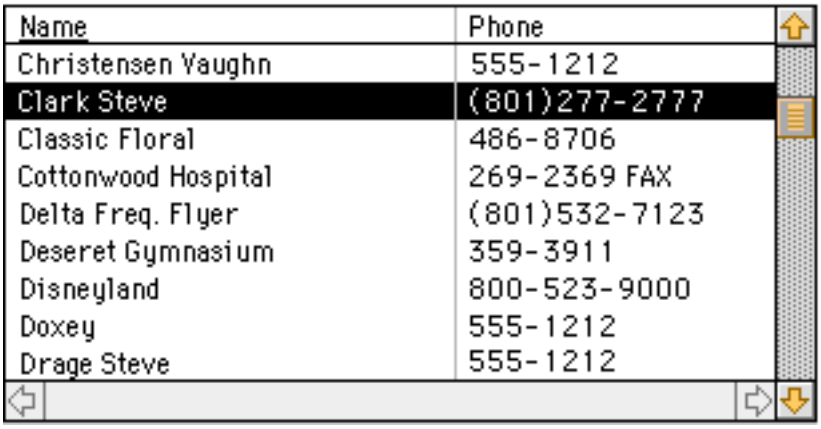
5. To further narrow down your entries, enter a keyword in the Find: field and press the Find button.

You may enter in a name, date, number, or other search criteria and click on Find or press return to initiate the search. "Find" searches all entries and fields in the currently selected application and in the current overview. All entries that match your search criteria—the search results—display in the overview. The status text alerts you if no search results are found.

The Find "magnifying glass" icon appears at the left while you are in Find mode. To clear this mode and return to the regular overview, press Find All button.

Understanding the Entry Overview

You have now specified the criteria for exporting your Newton entries. Now a few words about the overview: Once you make a connection or display a new application, Accordian automatically retrieves the overview—a table of your entries—and displays it on the right side of the screen.



Name	Phone
Christensen Vaughn	555-1212
Clark Steve	(801)277-2777
Classic Floral	486-8706
Cottonwood Hospital	269-2369 FAX
Delta Freq. Flyer	(801)532-7123
Deseret Gymnasium	359-3911
Disneyland	800-523-9000
Doxey	555-1212
Drage Steve	555-1212

Since your Newton application may contain hundreds or even thousands of entries, Accordian loads them in the overview only as you need them. As you scroll down the overview, Accordian fills in the entries from your Newton in blocks of 25. It takes a few seconds for the Newton to send up the entries and fill them in the overview, so you may see blank areas if you scroll fast enough.

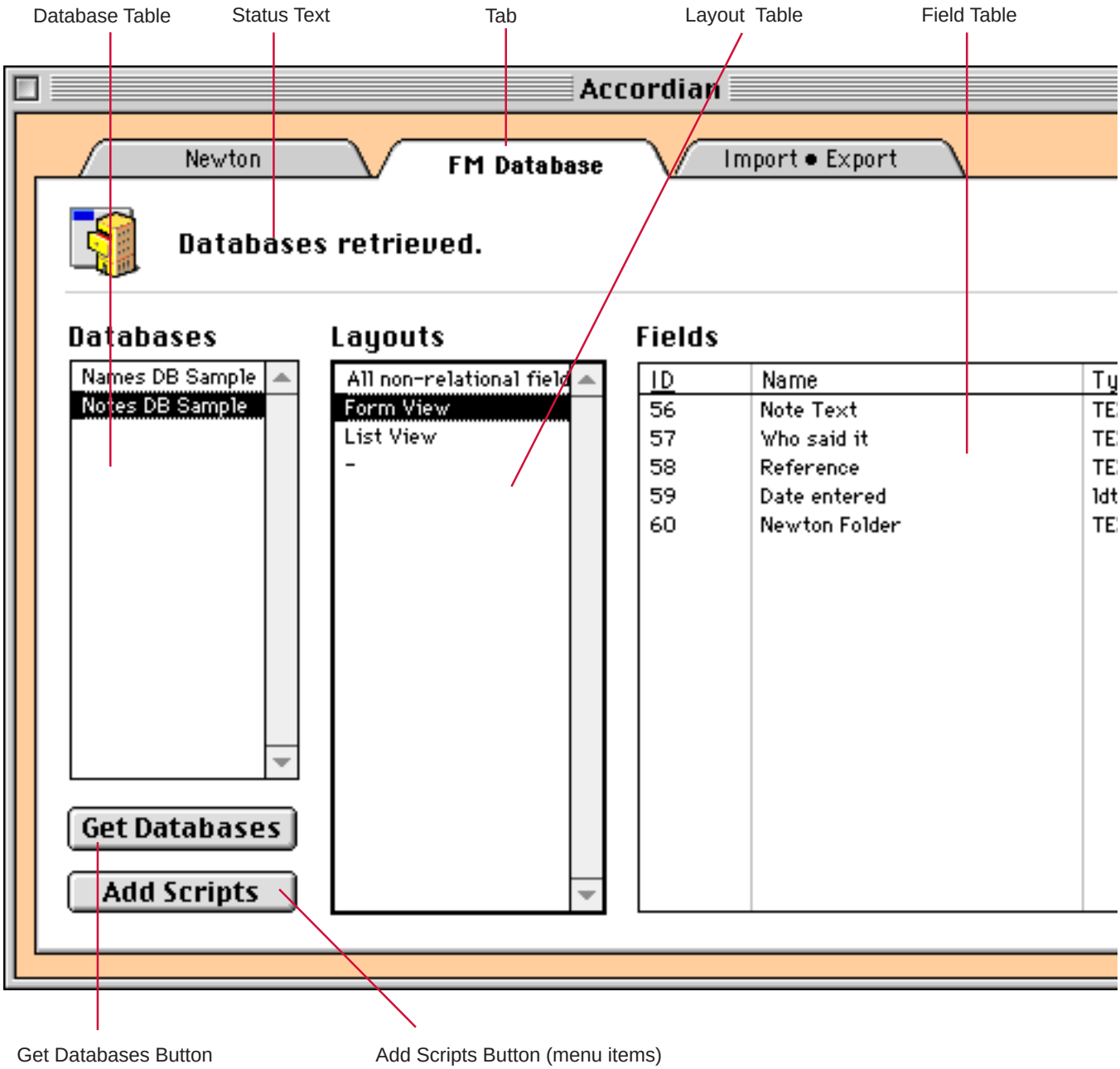
The overview reloads each time you: change the folder, perform a find, change the range, or change the application.

Locked Applications

An application is locked if at the left of the App: pop up menu there is a small padlock icon. A locked application or entry is read-only, which means you can export the contents but not import to it. The To Do application is a locked application.

FM Database Tab

The FM Database tab enables you to access the database you want to export to or import from. Because Accordian is a live link to your databases, you must have FileMaker Pro running in the background. You need not launch it before launching Accordian, but you can save some time by doing so, if you have Accordian automatically retrieve your databases when it launches.



Retrieving Database Information

Before going any further, make certain that you are using FileMaker Pro 3.x. If you attempt to use FileMaker 4.0, Accordian will be unable to retrieve any database information. Revelar is considering an update to Accordian that supports version 4.0.

1. Launch FileMaker Pro 3.0 along with your desired database file. As far as Accordian is concerned, you may open as many database files as you wish.

2. Make Accordian the front-most application and press the Get Databases button.

Accordian lists all of the open database files. The sample files shipped with Accordian are shown in our example screen shot on page 13.

3. Click on the desired database.

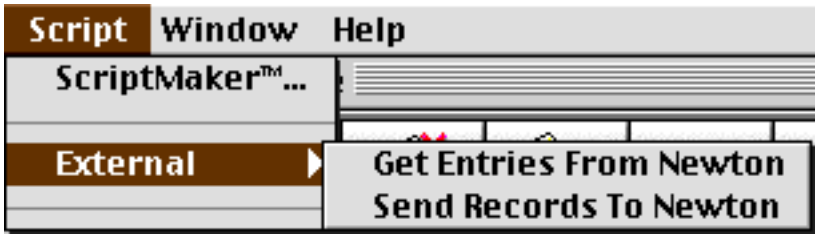
All the layouts created in that database appear in the Layout Table at the right.

4. Click on the desired layout.

All the fields displayed in that layout appear in the Field Table at the right. If you click on a field in the list, Accordian shows you what type of field it is in the Status Text. This may be helpful when you are matching or mapping fields in the next tab, since the Newton may require certain field types to be imported. You are now ready to move to the next tab.

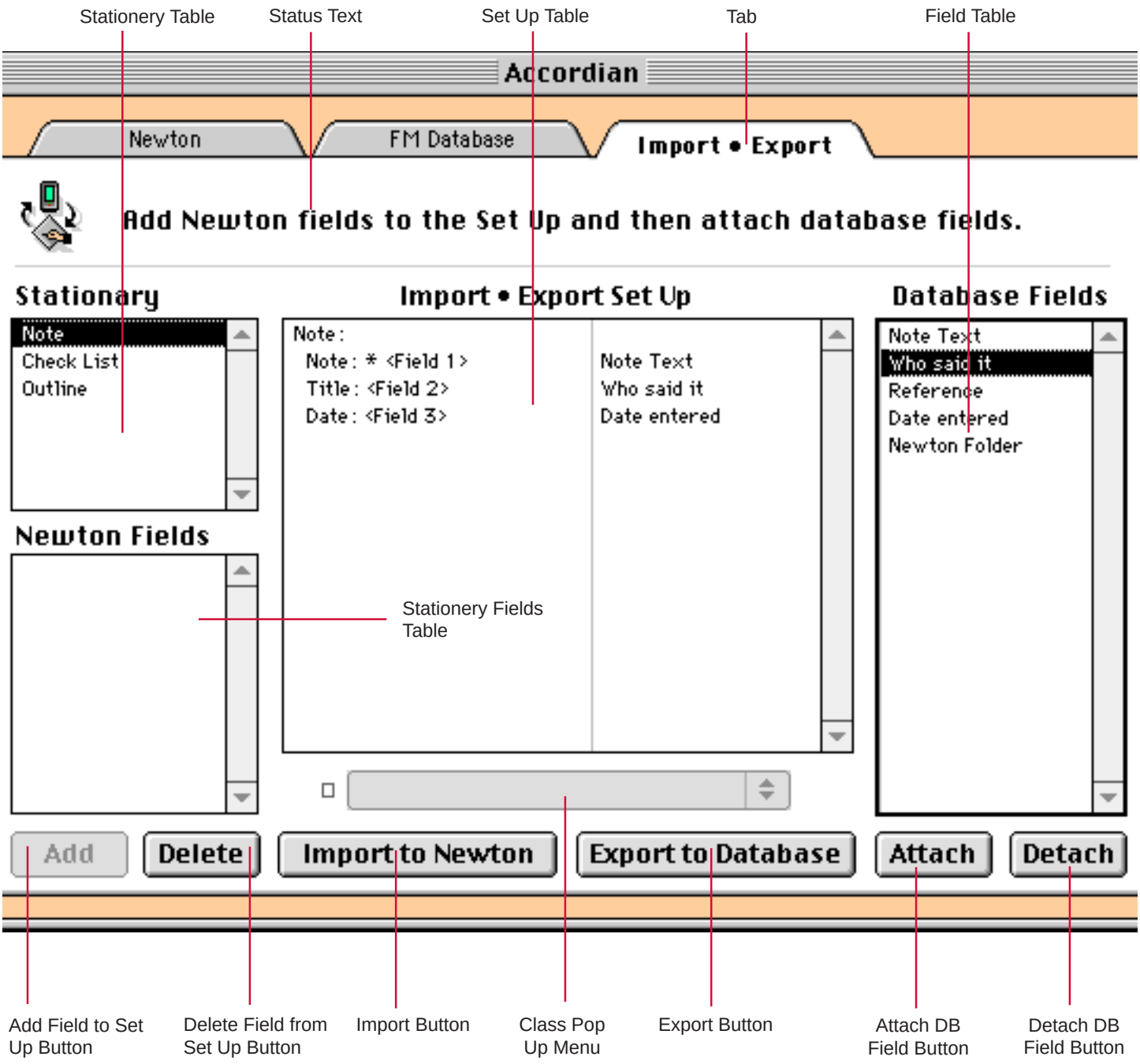
Adding Menu Item Scripts to FileMaker Pro

You can add menu items to FileMaker Pro that will allow you to control import and export directly from the database. How useful this feature is depends on the frequency and nature of your work. Press the Add Scripts button to add two menu items to FileMaker under the Scripts > External menu. If, for any reason, you force quit Accordian or have a crash, it is recommended that you quit FileMaker Pro and relaunch it along with Accordian before you use the menu items.



Import • Export Tab

This tab is where we actually match or "map" the fields between the Newton and the database. Once you have the minimally required Newton fields added to the Set Up and have made at least one attachment from the database, you will be able to import or export. The process is to first add Newton fields, and then attach the appropriate database fields to them in the Set Up table.



Matching or Mapping Fields

Since applications on the Newton can have different stationery types, you must choose, and therefore import or export, a single stationery. See your Newton PDA's user guide for more information on stationery.

1. Select the desired stationery type in the Stationery table, such as Checklist, Person, or Company.

Accordian lists the fields for that stationery in the Newton Fields table below and automatically adds to the Set Up table the first required fields. In the case of a person, for example, First and Last name are required (although Honorific is also added).

2. Click on the desired fields in the Newton Fields table and press the Add button to add them to the Set Up. This is done one at a time.

Fields are added in a required order by the ADF file. Advanced users may modify the order of these fields by making changes to the corresponding ADF file. Also, certain fields, such as address, add a group of fields to the Set Up. Unwanted fields may be deleted by selecting the row in the Set Up table and then pressing the Delete button.

Note: You may also add fields to the Set Up table by double-clicking the name in the Newton fields table.

3. Select a row in the Set Up table that you want to attach a field to and then select a row in the Database Fields table. Press the Attach button. Repeat this for each row in the Set Up table.

This process is called "mapping the fields." It means that the data in one field will be transferred to the field attached to it on the same row. The field name is placed in the column next to the Newton field. To remove the mapping of a certain database field, select the row and press the Detach button. If you make a mistake on a row that should be mapped with another database field, don't bother detaching it first. Just attach the correct field and Accordian will override what was previously put there.

If the field has a class type, such as a phone number, the class pop-up menu is active. For example, you specify the class of a phone number by selecting the appropriate pop-up menu item. The pop-up menu will activate when a field has a class type.

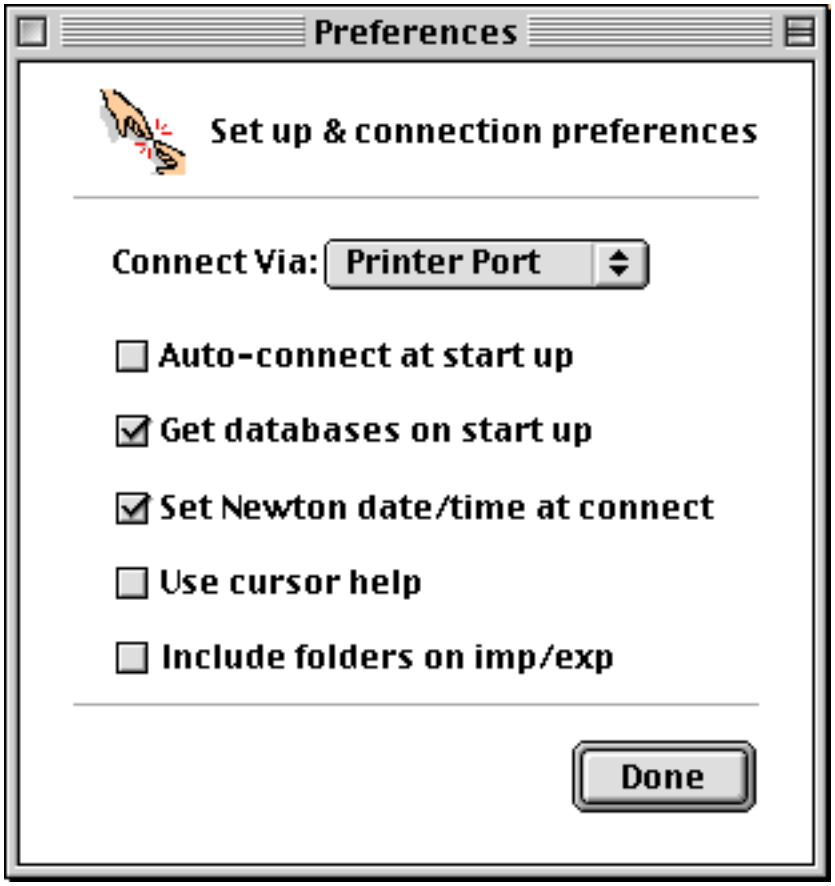
4. If you are importing, go to FileMaker and do a Find to select only those records you want to send to the Newton.

5. Once you are confident your Set Up is finished, press the Import to Newton or Export to Database button.

Accordian begins the import or export of the data. Import is roughly twice as fast as export, since export requires the Newton to evaluate each entry first to see if it is the correct stationery type.

Preferences

The Preferences window allows you to select different usage options.



Connect Via:

Specify which port Accordian should use when trying to make a connection to the Newton. You can select any available serial port in the pop up, but the current version of Accordian does not support AppleTalk.

Auto-connect at start up

Check on the Auto-connect at start up button if you want Accordian to automatically attempt to connect to the Newton device as soon as it launches. Checking this box only saves you the trouble of pressing Connect on the Macintosh side. You must still press the Connect button on the RCU Newton software.

Get databases at start up

Checking this box tells Accordian to retrieve open database files from FileMaker at start up. For this to be effective, FileMaker must be launched prior to Accordian.

Set Newton date/time at connect

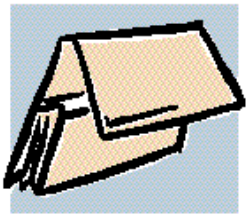
Checking this box synchronizes the date and time of your Newton with that of your Macintosh upon making a successful connection.

Use cursor help

Checking this box provides you with context-sensitive help in the Status text. This means that you can point the arrow cursor at any object in Accordian's window and brief help text will show up in the Status text.

Include folders on imp/exp

Checking this box lets you include Newton folder names in your data transfer. This means you could import FileMaker records into specific Newton folders. This checkbox is disabled if the currently selected Newton application does not support folders, such as Calendar.



Advanced Use

ADF File Format

The Newton 2.0 operating system provides new possibilities for extending the applications that run on the Newton platform. This makes Newton devices very powerful for both data acquisition and user customization.

This extensibility makes it very difficult to write a generalized desktop connectivity application because each Newton device could have different types of extensions, even extensions to“ built-in” applications. Because of the inability to“ hard code” application definitions into an application, Revelar decided to externalize this information in the form of an ADF file, or Application Definition Format file.

The format of this file is simple enough that non-programmers can understand it, yet powerful enough to describe almost any Newton data structure. It focuses on defining applications, soups, entries, and extensions.

Overview of the ADF File Format

You use the ADF file format to define entries in soups that exist on Newton devices. It describes most formats that can exist, including all built-in applications. It is also modular, meaning that each block of information builds on previous information, even if split across multiple files. This means that a single file can be used to describe the standard Newton application formats and other files can be included by third parties, each describing either a new application or an extension to a previously defined application.

Another powerful use of the ADF file is localization. Since most information presented to the user in Accordian comes from the ADF file, changing the language of the strings in the ADF file has the effect of localizing the Accordian application.

Because ADF files can refer to information in previously read files, the order in which ADF files are read is critical. For the Accordian application, the first ADF file read is the newton.adf file. If this file is not found, then a default version, coded into the program, is used. All other files in the folder are read in alphabetical order (of course the newton.adf file is skipped during this phase). All ADF files are read from a folder called ADF Folder that must exist in the same directory as the Accordian application. If the folder does not exist then only the built-in newton.adf file is used, giving access to only the built-in features of the Newton 2.0 applications.

General Format

ADF files are stored as text files. The standard suffix for these files is “.adf”. The files can be created with any standard text editor and are completely cross platform. Comments can be included in ADF files using the C++ standard, meaning that comments begin with two forward slash characters (//) and extend to the end of the line. In several places the syntax allows for a list of the same object. In this case an ellipsis (...) is used. Throughout this document certain items are referred to, including the following:

Item	Syntax	Example
string	" <any chars but return> "	"Names"
symbol	Alphabetic, then alphanumerics	| <any chars but return> |
		Slot9
		|a-b.5|
number	List of digits	136
path	Alphabetic, then alphanumerics	| <chars> .<chars> ... |
		Slot9
		|name.first|

There are seven top level objects that appear in ADF files: application, soup, overview, entry, class, range and extension. Each of these blocks is described in the following sections. Blocks cannot be split across files, but that blocks in one file can refer to blocks in a previously read file.

Application Syntax

application string called string with options { string ... }

Strings

The first string is the application symbol, a unique identifier for every Newton application. It must be obtained either from the author of the application or by inspection of the Newton data structures.

The second string is the user name for the application or the text that appears in Accordian.

Options In Braces

localfolders. The application doesn’t use global folders.

nofolders. The application doesn't support filing.

readonly. Application data can be viewed but not changed. This option allows export, but not import.

Soup Syntax

soup string called string of string range string with options { string ... }
metasoup string called string of string range string with options { string ... }

This block describes a soup. A single application may have more that one soup. Each application/soup pair appears in the Application menu of Accordian.

Strings

The first string is the soup name on the Newton. It can be obtained either from the author of the application that owns the soup or by inspection of the Newton data structures.

The second string is the user name for the soup, or the text that appears throughout Accordian.

The third string identifies the application that the soup belongs to. The string must be equal to the application symbol of an application block that has already been encountered.

The range information is optional. If given, the string refers to a range definition that will be provided later on in the file.

Metasoups

The second syntax defines a special object called a metasoup. A metasoup defines an object that is not a soup, but that can be treated like a soup by Accordian. Metasoups represent extensions to the RCU package on the Newton that provide an API that Accordian and RCU talk to by way of frames.

The most common example of a metasoup is the built-in Dates application on the Newton—the format of the entries in the application soups are either undocumented or too complex to map directly into Accordian (repeating meetings, etc.).

Options In Braces

readonly. Soup data can be viewed but not changed. This option allows export, but no import.

nofolders. The soup does not use folders, even if it belongs to an application that does support folders.

Overview Syntax

overview string called string of string queries string contains { column ... }
where column has the following syntax:
column string width number is path

This block defines an overview for a soup. Information in the overview window of Accordian allows the user to view arbitrary information in a tabular format.

Strings

The first string declares a symbol for the overview. It does not correspond to information on the Newton. It must be unique among all overview blocks in all ADF files.

The second string is the user name for the overview, which appears in the Overview palette in RCU, but does not appear in Accordian.

The third string identifies the soup for which the overview is valid, and is equal to the name of the soup (not the user name).

The fourth string identifies the slot that is indexed for the query for the overview. Normally it is the indexed slot for the soup for which the overview is valid.

Columns

The list of column information defines the columns that appear in the overview. The columns appear left to right in Accordian, in the same order that they appear in the list. Each column line defines the information for a column of the table.

The first string defines the user name of the column—the text appears as the header of the column. The number defines the width (in pixels) of the column. The path declares the path to the slot of the entry that appears in the column.

Note: Advanced users can edit the newton.adf file to create custom overviews (i.e., change the column names or display different information). See "Creating Custom Overviews" in this section for instructions. See "Understanding Overviews" in this guide for an introduction to overviews.

Entry Syntax

class string called string of string with options { string ... } contains { slot ... }
where slot is one of the following:
symbol : type called string class string default string with options { string ... }
symbol : set called string first symbol with options { string ... } [array contents ...]

```
symbol : entry called string with options { string ... } contains { slot ... }
symbol : class string called string with options { string ... } contains { slot ... }
symbol := symbol string
symbol := bounds { number number number number }
symbol := value number
symbol := nil
symbol := format string { path ... }
```

The Entry Syntax is the most complicated and powerful block in the ADF syntax. It defines the contents of an entry of a soup in terms of the slots and types of objects that exist in the entry.

Strings

The first string is the class of the entry. It must be obtained from the author of the application that owns the soup the entry appears in, or from the author of the extension. It is the value of the class slot for the entry.

The second string is the user name for this type of entry and it appears in several places in RCU and Accordian.

The third string defines the name of the soup that the entry belongs in (a reference to a soup also references an application because soups are attached to applications).

Optional strings are listed below the third string. The slot list defines the contents of the entry. Each of these lines begin with a symbol and the rest depends on the type of information stored. Because of the complex formats involved, each slot type and the options associated with it are described in the following sections.

Simple Slot

```
symbol :  type called string class string default string with options { string ... }
```

This syntax defines a simple slot with a given type:

boolean. A true/false value.

integer. A number, no floating point.

real. A floating point value.

string. A single line of text, no returns allowed.

text. Multiple lines of text, no graphics.

note. A Newton Note object, both text and graphics.

date. A date.

time. A time.

datetime. A single value storing both date and time.

The first string defines the user name of this slot. If the value has a class, then the class string is given. The class string in this case refers to a class block, described below, and not to a Newton class (although the class block does define the ultimate class the object will have). If a default value should be given to the object whenever it is added to an entry then it is listed.

Options strings can follow, and come from the following list, which applies for slots and for entries.

hidden. The value is invisible.

suffixfromclass. When the value is displayed, a suffix will be added based on the class the object has.

titlefromclass. When the value is displayed, the title of the object will be based on the class the object has. It will override the user name declared in the ADF file.

required. The slot must be present in order for the entry to be saved to the Newton. Note that this option implies the **automatic** option.

automatic. The slot will be added automatically whenever an entry is created.

readonly. The value can only be viewed, not edited. This will allow export, but not import of this slot.

displayifempty. The slot should be displayed even if it contains an empty value. The definition of Empty changes for each type. As an example, an empty string value would normally not be shown in Accordian. This option overrides this behavior.

firstclasswild. The first class defined for this slot is "wild", meaning that if the slot has a class other than those defined for it the system may use the first class definition. This is useful for things like phone number definitions, where the first class is generic.

outlinenote. Used on arrays slots that define the contents of "Outline" notes on the Newton. Used as a flag for import/export that the contents of the array should be translated to or from a textual format.

checklistnote. Used on arrays slots that define the contents of "Checklist" notes on the Newton. Used as a flag for import/export that the contents of the array should be translated to or from a textual format.

Array Slot

symbol : set called string first symbol with options { string ... } [array contents ...]

This syntax defines an set. The first string gives the user name of the set. The first value, if given, tells the system that the first setelement is stored in the root object and not in the array. It may either be a slot name, in which case the array contents should be single values (like strings), or it can be empty, in which case the array contents should be a frame. The way to specify an empty symbol is with a set of vertical bars (||). The options list is the same as for simple slots.

The set contents can either be another array definition, a frame definition, or a simple slot definition, but the symbol and colon are left off. See "Example ADF File" in this guide for clarification.

Note that sets are slightly different from arrays, which are not yet supported. In a set each element is interchangable with any other element, meaning there is no ordering to a set.

Entry Slot

symbol : entry called string with options { string ... } contains { slot ... }

This syntax defines a subframe of an entry or frame. The first string defines the user name of the slot. The options are the same as for a simple slot. The slot list is equivalent to the slot list for the entry block. Entry definitions are used frequently as array elements, where the array elements are unclassed frames of information.

Class Slot

symbol : class string called string with options { string ... } contains { slot ... }

Class slots allow a single slot to have multiple definitions, based on the class given the frame. They are equivalent to entry slots, but allow the same symbol to have multiple definitions. If a slot did have multiple definitions it would appear twice in the slot list, each time with a different class slot definition. Other than that difference, the values are the same as for an entry slot.

Computed Slots

symbol := symbol string
symbol := bounds { number number number number }
symbol := value number
symbol := nil

These definitions allow for a slot to take a constant value, and force the slot to be hidden. The first definition defines a symbol value to be stored in the slot. The second creates a bounds rectangle. The numbers are in the order of top, left, bottom, and right. The third definition allows for numeric values and the last creates a slot with a nil value.

Format Slots

symbol := format string { path ... }

This definition allows for computed fields—fields that depend on the values of other slots. The current syntax only allows for text slots to be used. The string gives the format used to compute the value and has the syntax of the format string of the sprintf function in the C language. The path list

gives paths to all slots used in the format. The following example computes the “sortOn” value in the “ Names” soup:

```
sortOn := format “%s %s” { name.last name.first }
```

Class Syntax

field class string symbol is string string

This block defines class information for a slot value. It is referred to by the class keyword in the slot definition. The value of that keyword must match the first string. Multiple class blocks may share the same class name (they usually will).

The class name does not have any significance on the Newton—it only serves to identify objects that can have these classes. The symbol given is the class that the object will have when this particular class is chosen, and the last two strings given the user name (or title value) for the class and the suffix value of the class.

For examples of the use of this block see the slot definition for a phone number in the Names soup (see “ Example ADF File”). Note that class blocks do not need to appear before they are referenced in an ADF file. Class blocks with the same name can appear in different files, they define new class possibilities.

Range Syntax

range string called string is string

This block defines information about ranges. The first string is the name for a set of related ranges and corresponds to the name used in the soup definition to refer to the range. The third string is the range definition.

The last string contains the definition of the range. Ranges define four pieces of information: Start closure, start element, last element and last closure. The start and last elements can either be thought of as part of the range or not. If an element is part of the set then it is "closed", otherwise it is "open". Examples of the types of ranges can be found in the same ADF file for the Names, Notes and Calendar applications. If an element is closed then a bracket is used to begin or end the string ('[' or ']'). If an element is open then a parenthesis is used to begin or end the string ('(' or ')'). The characters '...' (three periods with no spaces) must appear between the start and last element definitions.

There are four different types of range elements. Range elements may also be empty (not given), in which case the selection extends to the beginning or the end of the soup. Because of this the string "[...]" would be a global range, including in it every entry in the soup.

Date Range Elements

The first type of range element is a date. A date may be specified as a fixed date or a calculated date. Fixed dates are given as 'mm/dd/yy' or 'mm/dd/yyyy'. Computed dates are given as a keyword, optionally followed by a '+' or '-' and an offset. The keywords for computed dates are: **daystart**, **weekstart**, **monthstart** and **yearstart**. Each of these keywords refers to midnight at the start of the unit (daystart means midnight of the current day). Offsets are numbers followed by a suffix that gives the unit: **h** for hour, **d** for day, **w** for week or **y** for year.

Number Range Elements

Number range elements are given by putting a number for the element.

String Range Elements

String elements are given as strings, in double quote marks. **Note** that since range elements themselves appear in a string that these string range elements must have their double quote marks escaped by putting a '\' character in front of them. See the Names range definition for an example.

Extension Syntax

extension of string of string { slot ... }

Extension blocks allow extending the definition of an entry definition in another file, allowing third parties to ship small definition files that define just their extensions to other applications without having to modify the definition of the entry definition.

Strings

The first string references the class of the entry, while the second string is the name of the soup the entry belongs to. The slot definition list is equivalent to that used in the original definition of the entry.

Example ADF File

```
// Notes Application
application "paperroll" called "Notes"

// Notes Soup
soup "Notes" called "Notes" of "paperroll" range "optionalDateRange"

overview "notes:date" called "By Date" of "Notes" queries "timestamp" contains
```

```
        {
        column "Date" width 155 is timestamp
        column "Note Title" width 200 is title
        }

class "paperroll" called "Note" of "Notes" contains
        {
        data: note called "Note" with options { "required" "automatic" }
        title: string called "Title"
        timestamp: datetime called "Date" default ""
        viewstationary := symbol "paperroll"
        height := value 200
        }

class "checkList" called "Check List" of "Notes" contains
        {
        timestamp: datetime called "Date" default ""
        title: string called "Title"
        topics: array called "Topics" with options { "checklistnote" "required" } { entry called "Topic"
contains
                        {
                        mtgdone: boolean called "Checked" default "O"
                        hideCount: integer called "Hide Count" default "0"
                        level: integer called "Level" default "1"
                        text: text called "Text" with options { "automatic" }
                        viewbounds := bounds { 0 0 200 30 }
                        }
                }
        viewstationary := symbol "paperroll"
        height := value 200
        data := nil
        }

class "list" called "Outline" of "Notes" contains
        {
        timestamp: datetime called "Date" default ""
        title: string called "Title"
```

```
        topics: array called "Topics" with options { "outlinenote" "required" } { entry called "Topic"
contains
                        {
                        hideCount: integer called "Hide Count" default "0"
                        level: integer called "Level" default "1"
                        text: text called "Text" with options { "automatic" }
                        viewbounds := bounds { 0 0 200 30 }
                        }
                }
        viewstationary := symbol "paperroll"
        height := value 200
        data := nil
        }


// Names Application
application "cardfile" called "Names"

soup "Names" called "Names" of "cardfile" range "textRange"

overview "names:phones" called "Phone List" of { "cardfile" "Names" } queries "sortOn" contains
        {
        column "Name" width 150 is sortOn
        column "Phone" width 117 is [ phones, 0 ]
        }

overview "names:email" called "E-Mail List" of { "cardfile" "Names" } queries "sortOn" contains
        {
        column "Name" width 150 is sortOn
        column "E-Mail" width 117 is email
        }

overview "names:company" called "Company List" of { "cardfile" "Names" } queries "sortOn" contains
        {
        column "Name" width 150 is sortOn
        column "Company" width 117 is company
        }
```

```
overview "names:bday" called "Birthday List" of { "cardfile" "Names" } queries "sortOn" contains
        {
        column "Name" width 150 is sortOn
        column "Birthday" width 117 is bday
        }

class "Person" called "Person" of { "cardfile" "Names" } contains
        {
        sortOn := format "%s %s" { name.last name.first }
        name: entry called "Name" with options { "required" } contains
                {
                honorific: string called "Honorific" with options { "automatic" }
                first: string called "First" with options { "automatic" }
                last: string called "Last" with options { "automatic" }
                }

        addresses: set called "Address" firststoredin || { entry called "Address" contains
                        {
                        address: string called "Street"
                        address2: string called "Street"
                        city: string called "City"
                        region: string called "Region/State"
                        postal_code: string called "Postal Code"
                        country: string called "Country"
                        }
                }

        companies: set called "Company" firststoredin || { entry called "Company" contains
                        {
                        company: string called "Name"
                        title: string called "Title"
                        }
                }

        pagers: set called "Pager" firststoredin || { entry called "Pager" contains
                        {
```

```
                    pagerNum: string called "Pager" class "pagerClasses" with options { "titlefromclass" "firstclasswild" }
                    pagerPIN: string called "Pager PIN"
                    }
            }

        phones: set called "Phone" { string called "Phone" class "phoneClasses" with options { "titlefromclass" "firstclasswild" } }
        emailAddrs: set called "E-Mail" firststoredin || { entry called "E-Mail" contains
                    {
                    email: string called "EMail" class "mailClasses" with options { "titlefromclass" "firstclasswild" }
                    }
            }
        bday: date called "Birthday"
        anniversary: date called "Anniversary"
        notes: note called "Notes"
        cardType: integer called "Card Type"
        }


class "company" called "Company" of { "cardfile" "Names" } contains
        {
        sortOn := format "%s" { company }
        company: string called "Company" with options { "required" }
        names: set called "Name" { entry called "Name" contains
                    {
                    honorific: string called "Honorific"
                    first: string called "First Name"
                    last: string called "Last Name"
                    affiliation: string called "Title"
                    }
            }

        addresses: set called "Address" firststoredin || { entry called "Address" contains
                    {
                    address: string called "Street"
                    address2: string called "Street"
```

```
                    city: string called "City"
                    region: string called "Region"
                    postal_code: string called "Postal Code"
                    country: string called "Country"
                    }
            }

    phones: set called "Phone" { string called "Phone" class "phoneClasses" with options { "titlefromclass" "firstclasswild" } }
    emailAddrs: set called "E-Mail" firststoredin email { string called "EMail" class "mailClasses" with options { "titlefromclass" "firstclasswild" } }
    notes: note called "Notes"
    cardType: integer called "Card Type"
    }


class "owner" called "Owner" of { "cardfile" "Names" } with options { "nofolders" } contains
    {
    labels := symbol "_ownerNames"
    sortOn := format "%s %s" { name.last name.first }
    name: entry called "Name" with options { "required" } contains
            {
            honorific: string called "Honorific" with options { "automatic" }
            first: string called "First" with options { "automatic" }
            last: string called "Last" with options { "automatic" }
            }

    addresses: set called "Address" firststoredin ll { entry called "Address" contains
                    {
                    address: string called "Street"
                    address2: string called "Street"
                    city: string called "City"
                    region: string called "Region/State"
                    postal_code: string called "Postal Code"
                    country: string called "Country"
                    }
            }
```

```
companies: set called "Company" firststoredin || { entry called "Company" contains
                {
                company: string called "Name"
                title: string called "Title"
                }
        }

pagers: set called "Pager" firststoredin || { entry called "Pager" contains
                {
                pagerNum: string called "Pager" class "pagerClasses" with options { "titlefromclass" "firstclasswild" }
                pagerPIN: string called "Pager PIN"
                }
        }

phones: set called "Phone" { string called "Phone" class "phoneClasses" with options { "titlefromclass" "firstclasswild" } }
emailAddrs: set called "E-Mail" firststoredin || { entry called "E-Mail" contains
                {
                email: string called "EMail" class "mailClasses" with options { "titlefromclass" "firstclasswild" }
                emailpassword: string called "Password"
                }
        }

owner: entry called "Owner Information" contains
                {
                bankAccounts: set called "Bank Account" { entry called "Bank Account" contains
                                {
                                bankAcctNum: string called "Acct #"
                                bankContactNum: string called "Phone" class "phoneClasses" with options { "titlefromclass" "firstclasswild" }
                                }
                        }

                creditCards: set called "Credit Card" { entry called "Credit Card" contains
                                {
                                creditCardNum: string called "Card #"
```

```
                                        creditCardName: string called "Name" class "creditClasses" with
options { "titlefromclass" "firstclasswild" }
                                        creditCardExpDate: date called "Exp. Date"
                                        creditCardContactNum: string called "Phone #"
                                        }
                        }
                }

        bday: date called "Birthday"
        anniversary: date called "Anniversary"
        notes: note called "Notes"
        cardType: integer called "Card Type"
        }

class "worksite" called "Worksite" of { "cardfile" "Names" } with options { "nofolders" } contains
        {
        labels := symbol "_ownerNames"
        sortOn:= format "%s" { place }
        place: string called "Worksite" with options { "required" }
        areaCode: string called "Area Code"
        dialingPrefix: string called "Dialing Prefix"
        notes: note called "Notes"
        }

field class "creditClasses" lstring.cardl is "Card" ""
field class "creditClasses" lstring.card.phonecardl is "Phone Card" ""
field class "creditClasses" lstring.card.creditcardl is "Credit Card" ""
field class "creditClasses" lstring.card.phonecard.attl is "AT&T" ""
field class "creditClasses" lstring.card.phonecard.mcil is "MCI" ""
field class "creditClasses" lstring.card.phonecard.sprintl is "Sprint" ""
field class "creditClasses" lstring.card.creditcard.visal is "VISA" ""
field class "creditClasses" lstring.card.creditcard.mastercardl is "MasterCard" ""
field class "creditClasses" lstring.card.creditcard.amexl is "AmEx" ""
field class "creditClasses" lstring.card.creditcard.discoverl is "Discover" ""

field class "pagerClasses" lstring.pagerl is "Pager" ""
field class "pagerClasses" lstring.pager.skytell is "SkyTel" ""
```

```
field class "pagerClasses" |string.pager.mobilecomm| is "MobileComm" ""
field class "pagerClasses" |string.pager.embarc| is "EMBARC" ""

field class "phoneClasses" otherPhone is "Phone" ""
field class "phoneClasses" homePhone is "Home" "H"
field class "phoneClasses" workPhone is "Work" "W"
field class "phoneClasses" faxPhone is "Fax" "F"
field class "phoneClasses" carPhone is "Car" "A"
field class "phoneClasses" mobilePhone is "Cellular" "C"
field class "phoneClasses" homeFaxPhone is "Home Fax" "HF"

field class "mailClasses" |string.email| is "Email" ""
field class "mailClasses" |string.email.internet| is "Internet" ""
field class "mailClasses" |string.email.aol| is "America Online" ""
field class "mailClasses" |string.email.compuserve| is "CompuServe" ""
field class "mailClasses" |string.email.mcimail| is "MCI Mail" ""
field class "mailClasses" |string.email.attmail| is "AT&T Mail" ""
field class "mailClasses" |string.email.easylink| is "EasyLink" ""
field class "mailClasses" |string.email.prodigy| is "Prodigy" ""
field class "mailClasses" |string.email.genie| is "GEnie" ""
field class "mailClasses" |string.email.delphi| is "Delphi" ""
field class "mailClasses" |string.email.msn| is "Network" ""
field class "mailClasses" |string.email.interchange| is "Interchange" ""
field class "mailClasses" |string.email.radiomail| is "RadioMail" ""


// Calendar Application
// Note that the calendar application is written using metasoups as the
// format is too complex and not really documented.
application "calendar" called "Calendar" with options { "nofolders" }

// Calendar Metasoup
metasoup "calendar:metasoup:REVELAR" called "Meetings" of "calendar" range "dateRange"

overview "meetings:date" called "By Date" of "calendar:metasoup:REVELAR" queries "mtgStartDate"
contains
        {
```

```
        column "Date" width 155 is mtgStartDate
        column "Description" width 200 is mtgText
        }

class "meeting" called "Meeting" of "calendar:metasoup:REVELAR" contains
        {
        mtgStartDate: datetime called "Date" with options { "required" }
        mtgText: string called "Title" with options { "required" }
        mtgDuration: integer called "Duration" with options { "required" }
        notes: note called "Notes"
        mtgAlarm: datetime called "Alarm"
        location: string called "Location" with options { "readonly" }
        invitees: set called "Invitees" with options { "readonly" }
                {
                string called "Invitee" with options { "readonly" }
                }
        }

class "repeatingMeeting" called "Repeating Meeting" of "calendar:metasoup:REVELAR" contains
        {
        mtgStartDate: datetime called "Date" with options { "required" }
        mtgText: string called "Title" with options { "required" }
        mtgDuration: integer called "Duration" with options { "required" }
        notes: note called "Notes"
        mtgAlarm: datetime called "Alarm"
        location: string called "Location" with options { "readonly" }
        invitees: set called "Invitees" with options { "readonly" }
                {
                string called "Invitee" with options { "readonly" }
                }
        mtgStopDate: date called "Stop Date"
        repeats: string called "Repeats" class "repeatClasses" with options { "suffixfromclass"
"displayifempty" "required" }
        }

class "CribNote" called "Event" of "calendar:metasoup:REVELAR" contains
        {
```

```
        mtgStartDate: date called "Date" with options { "required" }
        mtgText: string called "Title" with options { "required" }
        notes: note called "Notes"
        mtgAlarm: integer called "Days Notice"
        }

class "repeatingCribNote" called "Repeating Event" of "calendar:metasoup:REVELAR" contains
        {
        mtgStartDate: date called "Date" with options { "required" }
        mtgText: string called "Title" with options { "required" }
        notes: note called "Notes"
        mtgStopDate: date called "Stop Date"
        mtgAlarm: integer called "Days Notice"
        mtgStopDate: date called "Stop Date"
        repeats: string called "Repeats" class "repeatClasses" with options { "suffixfromclass" "displayifempty" "required" }
        }

field class "repeatClasses" |daily| is "Every Day" "Every Day"
field class "repeatClasses" |weekly| is "Every week" "Every week"
field class "repeatClasses" |biweekly| is "Every other week" "Every other week"
field class "repeatClasses" |monthly| is "Every month" "Every month"
field class "repeatClasses" |monthlyByWeek| is "Same week each month" "Same week each month"
field class "repeatClasses" |yearly| is "Every year" "Every year"
field class "repeatClasses" |yearlyByWeek| is "Same week each year" "Same week each year"
field class "repeatClasses" |other| is "Other" ""

// To Do Application
application "todo" called "To Do" with options { "nofolders" "readonly" }

// To Do Soup
soup "To Do List" called "To Do List" of "todo" range "dateRange"

overview "todo:date" called "By Date" of "To Do List" queries "date" contains
        {
        column "Date" width 155 is date
        }
```

```
class "todo" called "To Do Item" of "To Do List" with options { "readonly" } contains
        {
        ldatel: date called "Date"
        topics: set called "Items" { entry called "To Do" contains
                        {
                        mtgdone: boolean called "Completed"
                        mtgPriority: integer called "Priority"
                        text: text called "Text"
                        }
                }
        }

range "dateRange" called "Today" is "<range> [ daystart ... daystart + 1d )"
range "dateRange" called "This Week" is "<range> [ weekstart ... weekstart + 1w )"
range "dateRange" called "Next Week" is "<range> [ weekstart + 1w ... weekstart + 2w )"
range "dateRange" called "This Month" is "<range> [ monthstart ... monthstart + 1m )"
range "dateRange" called "Next 30 Days" is "<range> [ daystart ... daystart + 30d )"
range "dateRange" called "Next 60 Days" is "<range> [ daystart ... daystart + 60d )"
range "dateRange" called "Previous 30 Days" is "<range> [ daystart - 30d ... daystart )"

range "optionalDateRange" called "All" is "<range> [ ... ]"
range "optionalDateRange" called "Today" is "<range> [ daystart ... daystart + 1d )"
range "optionalDateRange" called "This Week" is "<range> [ weekstart ... weekstart + 1w )"
range "optionalDateRange" called "Next Week" is "<range> [ weekstart + 1w ... weekstart + 2w )"
range "optionalDateRange" called "This Month" is "<range> [ monthstart ... monthstart + 1m )"
range "optionalDateRange" called "Next 30 Days" is "<range> [ daystart ... daystart + 30d )"
range "optionalDateRange" called "Next 60 Days" is "<range> [ daystart ... daystart + 60d )"
range "optionalDateRange" called "Previous 30 Days" is "<range> [ daystart - 30d ... daystart )"

range "textRange" called "All" is "<range> [ ... ]"
range "textRange" called "#..D" is "<range> [ ... \"E\" )"
range "textRange" called "E..H" is "<range> [ \"E\" ... \"I\" )"
range "textRange" called "I..L" is "<range> [ \"I\" ... \"M\" )"
range "textRange" called "M..P" is "<range> [ \"M\" ... \"Q\" )"
range "textRange" called "Q..T" is "<range> [ \"Q\" ... \"U\" )"
range "textRange" called "U..X" is "<range> [ \"U\" ... \"Y\" )"
```

```
range "textRange" called "Y..Z" is "<range> [ \"Y\" ... ]"

// Calls Application
application "callapp" called "Calls" with options { "readonly" }

// Calls Soup
soup "Calls" called "Calls" of "callapp"

overview "calls:date" called "By Date" of "Calls" queries "timestamp"
contains
        {
        column "Date" width 150 is timestamp
        column "Title" width 150 is title
        }

class "calllog" called "Call" of "Calls" contains
        {
        timestamp: datetime called "Date"
        title: string called "Title"
        name: string called "Name"
        phoneNumber: string called "Number"
        notes: note called "History"
        }
```

Customizing Accordian

You can create custom overviews in Accordian by editing the newton.adf file. You can also change the order of fields, which may be helpful if you want to import an existing document into Accordian (e.g., a spreadsheet file of names, addresses, phone numbers, etc.) and you need to change the default order of fields.

Important: Before making any changes the newton.adf file, save a backup copy of the original file.

Creating Custom Overviews

It is easy to edit the newton.adf file if you want to change the names of overview columns (e.g., edit "Phone" to read as "Telephone") or display new information. All changes are made in the newton.adf file. To create custom overviews:

1. Open the newton.adf file with any text editor (e.g., SimpleText).

2. Find the syntax for the application for which you want to create a custom overview.

The name of each application is preceded by //.

3. Find the string that starts with "overview."

For example, the Names application has two overview strings:

```
overview "names:phones" called "Phone List" of { "cardfile" "Names" } queries "sortOn" contains
        {column "Name" width 150 is sortOn
        column "Phone" width 110 is [ phones, 0 ]}
overview "names:email" called "E-Mail List" of { "cardfile" "Names" } queries "sortOn" contains
        {column "Name" width 150 is sortOn
        column "E-Mail" width 110 is email}
```

4. Edit the column name(s) or add a new overview.

To make editing changes, type over existing text. For example, you could change “ Phone” to “Telephone.”

5. Save the newton.adf file as a text file after making changes.

Changing the Field Order

You can edit the newton.adf file to change the order in which fields appear in an entry. This is an option if you are importing an existing document and the information (fields) for each entry are arranged in a different order.

For example, you may want to import a document with 1,000 lines of names (entries), with addresses, phone numbers, etc. (fields) into the Names application. To import this document correctly, the field information must be in the same order as the fields for the Names application. The default order of fields is set in the newton.adf file. Some users find it faster and easier to change the order of fields in the newton.adf file, rather than change the fields in the import document. To change the field order:

1. Open the newton.adf file with any text editor (e.g., SimpleText).

2. Find the syntax for the application for which you want to create a custom overview.

The name of each application is preceded by //.

3. Find the type of entry, called the class, that you want to edit fields in.

The string for each type of entry begins with "class." Fields are listed under each class.

For example, the class string and corresponding fields for the entry type "company" in the Names application is:

```
class "company" called "Company" of { "cardfile" "Names" } contains
        {sortOn := format "%s" { company }
        company: string called "Company"
        names: array called "Name" { entry called "Name" contains
                        {honorific: string called "Honorific"
                        first: string called "First Name"
                        last: string called "Last Name"
                        affiliation: string called "Title"}}
        addresses: array called "Address" firststoredin ll { entry called "Address" contains
                        {address: string called "Street"
                        address2: string called "Street"
                        city: string called "City"
                        region: string called "Region"
                        postal_code: string called "Postal Code"
                        country: string called "Country"}}
        phones: array called "Phone" { string called "Phone" class "phoneClasses" with options { "titlefromclass" } }
        emailAddrs: array called "E-Mail" firststoredin email { string called "EMail" class "mailClasses" with options { "titlefromclass" } }
        notes: note called "Notes"
        cardType: integer called "Card Type"}
```

There are six fields for the class type company: Name, Address (with sub fields Street, City, Region, Postal Code, Country), Phone, E-Mail, Notes, and Card Type.

4. Cut and paste each field block to change the order of fields.

For example, if you wanted phone numbers to be listed before address(es), then you would change the class string and field order as follows:

```
class "company" called "Company" of { "cardfile" "Names" } contains
        {sortOn := format "%s" { company }
        company: string called "Company"
        names: array called "Name" { entry called "Name" contains
                        {honorific: string called "Honorific"
                        first: string called "First Name"
                        last: string called "Last Name"
```

affiliation: string called "Title"}}

phones: array called "Phone" { string called "Phone" class "phoneClasses" with options { "titlefromclass" } }

addresses: array called "Address" firststoredin ll { entry called "Address" contains

{address: string called "Street"

address2: string called "Street"

city: string called "City"

region: string called "Region"

postal_code: string called "Postal Code"

country: string called "Country"}}

5. Save the newton.adf file as a text file after making changes.