APPENDIX A NOTEFILE CONCEPTS

This document provides a brief explanation of the structure of notefiles. It also describes how checkpointing, aborting, and recovering a notefile after a crash work. Finally, it describes the use of **Inspect & Repair** to repair a notefile with inconsistent links. For additional information see Appendix B, The Notefile Inspector.

Note: The information in this appendix was copied whole from the documentation for a prerelease of NoteCards and has not yet been updated for this release. As such, it may be inaccurate in places.

Background Concepts

Unique identifiers: UIDs

All objects in NoteCards (i.e., cards, links, notefiles) are assigned a unique identifier called a UID. Each UID is a 112-bit number that is guaranteed to be unique across all time and space. UIDs are used in many places in NoteCards as keys for indexing and retrieving cards, links, and notefiles.

Card parts

For storage purposes a note card is decomposed into 4 independent parts: contents, title, property list, and links. Each of these parts is stored separately in the data area of the notefile. This is discussed in the Notefile Structure section below. When a card is saved, only those card parts that have changed are rewritten in the notefile.

The contents of the card are stored on the notefile in a manner appropriate to its type. Thus a Text card's content is a text stream and is written on the notefile exactly the way TEdit writes out text streams (i.e., text followed by "looks" information). In contrast, all titles are stored as strings and all property lists as standard Lisp lists. Storage of Links is described in Section 4 below.

Notefile Structure: index and data areas

A notefile consists of three parts, a header, an index, and a data area. The header and the index are fixed in size for each notefile. The data area follows the index area and grows as cards are added to or modified in the notefile.

The notefile header contains the following information about the notefile: its UID, a number identifying its format, the checkpoint pointer, the size of the index, and a pointer to the next available index entry. If the notefile header is destroyed, it cannot be automatically reconstructed. Careful hand manipulation of the

notefile by a NoteCards wizard is required to recover a notefile with a bad header.

The structure of the index and data area is shown in Figure A-1 .

The index contains a fixed number of index entries. Each index entry that is in use, contains information for one of the cards in the notefile. Specifically, an index entry contains 5 fields: a status character, the card's UID, and 4 pointers. The status character specifies whether the index entry is free (not in use) or contains information for an active or a deleted card. If the index entry is not free, the UID field contains the UID of the card refered to by this index entry and the four pointer fields contain the location in the data area of the 4 parts of its card: contents, title, links and property list.

The number of index entries is fixed at notefile creation time. The default is 1000 entries. The number of index entries is automatically doubled by the notefile compactor if 75% of the entries are used. The compactor also frees (i.e., makes unused) all of the index entries that refer to deleted cards. In normal operation, NoteCards prints a warning whenever more than 90% of the index entries in a notefile are used. At this point, the notefile should be compacted to increase the index size.

The data area contains the actual information about the card. Whenever you change, say, the title of a card, the new title is written at the end of the data area. The title pointer in the card's index entry is then updated to point to this new location in the data area. Thus, in general, a notefile's data area grows every time any part of any card is changed.

The old information, now somewhere in the middle of the data area, is not removed. However, it is no longer directly accessible because there is no index entry that points to it. Thus, for most purposes this old information can be considered "dead space" in the notefile. The notefile compactor rewrites the notefile, eliminating all such dead space.

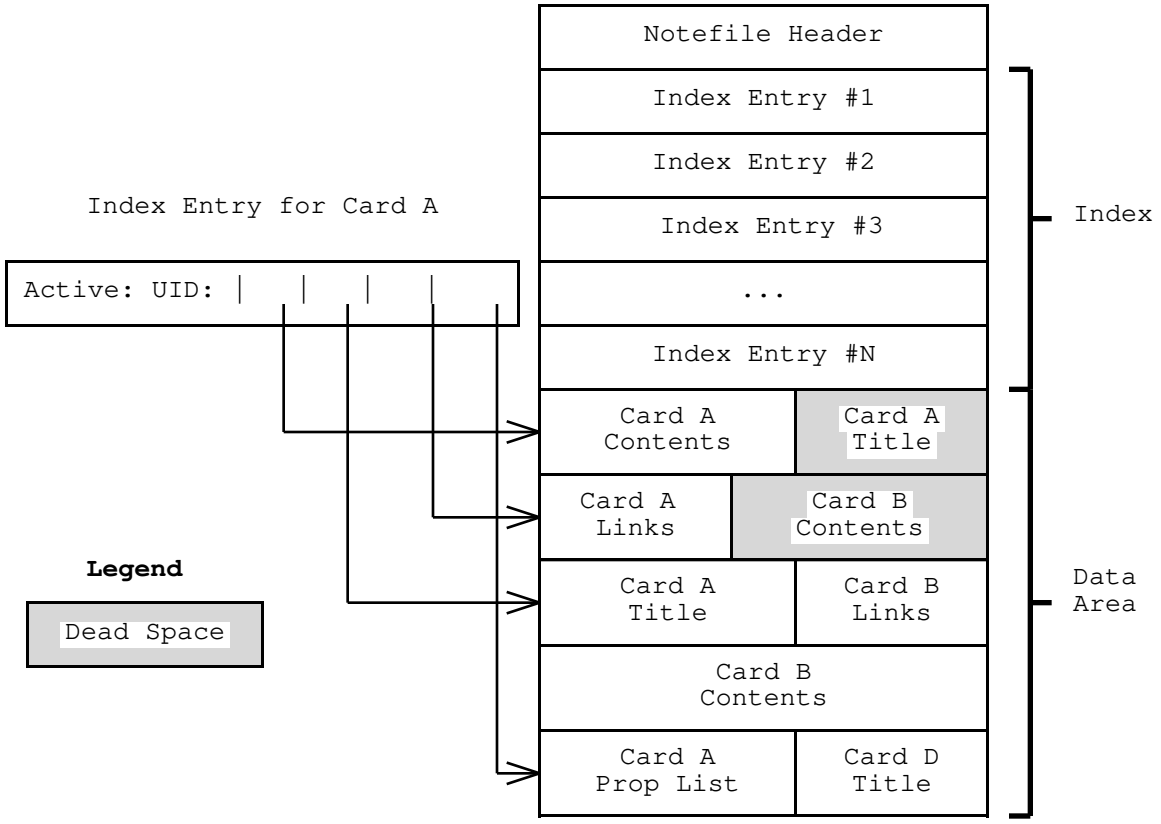

Figure A-1. The structure of a notefile.

As long as the notefile has not been compacted, all the old information can be accessed (and made to be "current") via the notefile Inspect and Repair facility. Inspect and Repair does this by ignoring the index and parsing the entire data area to produce a listing of all the information (both current and old) about a card that is stored on the data area. See the Inspect and Repair manual for more information.

Notefile Checkpointing

As long as a notefile is open, its index area is cached in memory. When a card part is saved, the card part's information is written to the end of the data area but the card's actual index entry on the notefile is not updated with this new location. Instead, the appropriate pointer in the card's in-memory index entry is updated. Thus, the index in the notefile continues to point to the old information in the data area, while the in-memory index points to the new information.

Thus while a notefile is open, its current state is distributed between the actual notefile and information cached in memory. The current index is cached in memory. For cards open on the screen (or cached in memory from the programmer's interface), the "current" card part information is contained in an in-memory cache. For all

other cards, the current information is contained in the data area of the notefile pointed to by the in-memory index entry.

If a machine crashes while a notefile is open, the information cached in memory is lost. A crash not only discards changes made to cards on the screen, but its also leaves the information stored on the notefile in an inconsistent state. For example, the index on the notefile may point to old information in the data area. This occurs because the new information (e.g., a new title) is written to the data area but only the in-memory index pointers (which are lost in the crash) have been updated to point to the new information.

Checkpointing forces all of the in-memory information to be written onto the notefile. Specifically, checkpointing causes all open cards to be saved and the in-memory index to be written to the notefile's index area. Thus, immediately after checkpointing, the notefile itself contains its current (and consistent) state. If the machine were to crash at this point, no information would be lost and the notefile would be consistent.

Checkpointing also writes a **checkpoint pointer** onto the notefile header. The checkpoint pointer contains the location of the end of the data area (i.e., the end of the notefile) at the time the checkpoint is done.

As the notefile is used after the checkpoint, information is written in the data area past the checkpoint pointer but only the in-memory index entries are updated to point to this information. The on-file index entries still point to the information in the data area *before* the location referenced by the checkpoint pointer. Thus, a consistent notefile can be constructed from the index area and all of the information in the data area located before the checkpoint location. This is essentially the notefile as it was at the time of the last checkpoint. (Note: one small exception is that changes to a card's size on the screen are actually written in the middle of the data area rather than at the end. Thus, truncating a notefile to its checkpoint location cannot "undo" the reshaping of a card.)

When opening a notefile after a crash, the system will insure that the notefile is in a consistent state. It does so by truncating the data area to the last checkpoint location, saving the truncated information if requested by the user. This leaves the notefile in the state it was during the last checkpoint before the crash.

Aborting a notefile does the same thing. It truncates the data area to the last checkpoint location, thereby eliminating all changes made to the notefile since the last checkpoint. It also discards the in-memory index. Thus, the notefile is left in the exact state it was after the last checkpoint.

Finally, note that notefile Close forces a checkpoint. Therefore, aborts and recovery after crashes actually restore the NoteFIle to its state as of the last user-initiated checkpoint *or* close.

Storing links and reparing notefiles with inconsistent links

The links card part is divided into three subcomponents: to-links, from-links, and global links. The to-links is a list of all links whose Source is the given card (i.e., that point from the card to some other card). The from-links is a list of links whose Destination is the given card (i.e., that point from some other card to the given card). Finally, the global links is a subset of the to-links that includes only the Global links originating in the given card.

Given this scheme, every link is stored on the notefile in three different places. First, if the link is Local it is stored inside the link icon which in turn is inside the content part of the link's source card. If the link is global, it is stored in the global links subpart of its source card. Second, the link is stored in the to-links list of its source card. Third, the link is stored in the from-links list of its destination card.

These three records of the same link occasionally get out of synch, resulting in an inconsistent notefile. There are a number of symptoms of such inconsistency. For example, the **ShowLinks** display for card may indicate that the card is a destination for a link from some source card X while the ShowLinks display for X does not include a ToLink to that destination card. Occasionally, inconsistent links will also result in link icons that contain the words "DELETED" or "FREE" when displayed on the screen. This usually means that the card at one end of a link was deleted, but somehow the links of the card at the other end were not updated. Such link icons cause NoteCards to break when you try to follow them.

One function of the NoteCards **Inspect & Repair** facility is to resynchronize the three records for all links in the notefile. The inspector's third phase rebuilds the links as follows. First it removes all to- and from-links for every card. Then it reads the contents for each card and recreates to-links and from-links by looking at the links found inside the link icons in the card's content and in its global links list. In addition, the links rebuilder phase of the notefile inspector can rebuild filebox contents from cards pointed to by the filebox, and the set of all link types from the list of all links.

[This page intentionally left blank]