amigaguide.doc

COLLABORATORS

	TITLE : amigaguide.doc	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		September 19, 2022	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

amigaguide.doc

1.1 AmigaGuide.doc

AmigaGuide TM

David N. Junod

1.2 COPYRIGHT

1.3 INTRODUCTION

The standard Amiga keyboard sports a HELP key, yet there has been no system provided support for this key. Now, there is AmigaGuide, which provides a standard method of displaying help and other on-line documentation to the user.

1.4 Capabilities

AmigaGuide uses an Intuition window that contains a scroll bar, buttons and pull-down menus, to display plain ASCII text files or AmigaGuide databases.

An AmigaGuide database is a set of related documents contained in one file. Each document may contain references to other documents, using what is called a link. A document may contain any number of links, pointing to any number of other documents. When the user selects a link, the document that the link points to will be displayed. The user may then use the links to read through the database, following whatever path they may choose. The technical term for AmigaGuide's abilities, is hypertext.

The user may at any time print a document or a portion of the document. They may also send portions of a document to the clipboard, for use in other applications.

Using ARexx, the user may write scripts, or an application could provide scripts, to control AmigaGuide.

Cross-reference tables can be loaded that specify where a keyword, or phrase is defined. The user can then use AmigaGuide's Find Document facility to quickly display a document based on keyword, without having to know the name of the database that it is located in.

AmigaGuide provides a unique feature to hypertext systems, called Dynamic Nodes. A Dynamic Node is a hypertext or plain text document that is generated in real-time as opposed to coming from a static file. An application that generates Dynamic Nodes is called a Dynamic Node Host.

1.5 Interfacing

AmigaGuide databases are accessed in three different ways:

- Databases can be browsed directly from the Workbench or Shell using a utility named AmigaGuide, a hypertext replacement for More.

- AmigaGuide support can be added to an existing application, that supports ARexx, by using AmigaGuide's ARexx function host capabilities.

- Applications can use the functions of AmigaGuide to provide help on gadgets, menus and windows. For example, the user could position the pointer over any gadget or menu item, press help, and the appropriate document would be displayed in the AmigaGuide window. The application could also have AmigaGuide display a pertinent portion of the current project.

1.6 Other Uses

In addition to help or on-line documentation, AmigaGuide has other possible uses. For example,

Tutorials
An application that has an ARexx port and supports AmigaGuide could set up a help system that not only provides help, but could also give examples. The user could read about a feature, then click on the EXAMPLE button, which would run an ARexx script that would give an example of use. For instances, to show Pattern Fill, the script could draw a circle, select a pattern, and then fill the circle.

Computer Aided Instruction
The student could read about different topics, following links. A multiple choice quiz could be set up at the end where the questions and answers run ARexx scripts to accumulate the score.

Program by Query
Many programmers develop using a Cut & Paste technique. They clip modules from the various applications or utilities that they have written, and paste them together to build new applications. A database could be set up of all these different code fragments (such as loading and saving ILBM's, clipboard access, playing sounds, etc.), you could step through, answering questions, while the sections that you need are being appended to a new source file.

1.7 USING AmigaGuide FROM WORKBENCH OR CLI

A number of utilities are provided to access AmigaGuide databases from the Workbench or Shell. These utilities are:

AmigaGuide

This tool is much like the system utility, More, but is also capable of
interpreting AmigaGuide databases.

LoadXRef
Used to load a cross-reference from disk into memory. Multiple files can be loaded.

ExpungeXRef
Used to clear the cross-reference table from memory.

1.8 AmigaGuide

The AmigaGuide utility is used for browsing through AmigaGuide databases. It can be run from the Workbench or the Shell.

To use it from Workbench, set the default tool of an AmigaGuide database's icon to AmigaGuide. Currently, the AmigaGuide utility doesn't recognize any tool type parameters.

From the Shell, AmigaGuide uses the following command template.

DataBase,Document/K,Line/N,PubScreen/K,PortName/K:

The following are descriptions of the arguments.

DataBase
The name of the AmigaGuide database to display.

AmigaGuide will look in the current directory for the database. If the database isn't found, then AmigaGuide will search through its path for the database. To set the path, see the
Path
paragraph in the

USER PREFERENCES
section.

Document
The document within the database to display. AmigaGuide will use the cross-reference table to automatically supply the DataBase and Line parameters.

Line
The line of the document to start displaying from.

PubScreen
The public screen to open on. Remember that public screen names are case sensitive.

PortName
The name to assign to the ARexx port for this occurence of AmigaGuide.

1.9 LoadXRef

The LoadXRef utility can be used from Workbench or the Shell to load cross-reference files from disk into memory. Multiple cross-reference files may be loaded at a time.

From Workbench, just set the default tool of a cross-reference file's project icon to LoadXRef. This tool doesn't support any tool type parameters.

From the Shell, just specify the name of the cross-reference file to load. LoadXRef will look in the current directory for the file. If the file isn't found, then LoadXRef will search through the users preference path for the file.

1.10 ExpungeXRef

The ExpungeXRef utility is used to remove all entries from the cross-reference table in memory.

1.11 USER PREFERENCES

AmigaGuide allows a number of items to be tailored to the users' preference. These preference items are stored in environment variables. The AmigaDOS command SetEnv can be used to set any of these variables.

In order to set any of the following environment variables, an ENV:AmigaGuide directory must be made.

makedir ENV:AmigaGuide

A Preference Editor that sets the AmigaGuide preferences would write in the ENV:AmigaGuide directory when "Use is selected, and write in the ENVARC:AmigaGuide directory when "Save" is selected.

Following is a list of the variable names, and what they control.

Path

Text

Pens

1.12 Path

This variable contains the list of directory names that AmigaGuide will search through when it attempts to open a database. The directory names are separated by a space.

For example:

```
SetEnv AmigaGuide/Path "Workbench:Autodocs Workbench:Includes"
```

1.13 Text

Used to specify the graphical style that the links are presented in. The possible styles are:

```
BUTTON  Draw a raised border around the text (default).
UNDERLINE Underline the text.
BOLD  Bold the text.
ITALIC  Italicize the text.
```

For example:

```
SetEnv AmigaGuide/Text BUTTON
```

1.14 Pens

This variable provides the user with the ability to specify the colors to use for the various renderings that AmigaGuide performs.

```
SetEnv AmigaGuide/Pens <abcdefgh>
```

Where:

```
a = Background pen
b = Button text pen
c = Button background pen
d = Highlighted button text pen
e = Highlighted button background pen
f = Outline pen
g = Highlight outline pen
h = Text on background pen
```

For example:

```
SetEnv AmigaGuide/Pens 21213001
SetEnv AmigaGuide/Text BOLD
```

Internally, AmigaGuide subtracts '0' from the pen number, so values can range from 0 to 207.

1.15 AUTHORING AmigaGuide DOCUMENTS

Authoring an AmigaGuide database, or any hypertext database for ↩ that
matter, is a difficult task. It takes a lot of insight into the
subject matter and how the pieces relate to each other. A database
must consist of documents that relate to each, documents must be
broken up into manageable chunks, and links must be carefully thought
out. A document should consist of information dealing with one
topic. Each document should contain links to other related documents.

An AmigaGuide database is ASCII text with embedded commands that tell
AmigaGuide how to interpret the database. A database should consist
of a main table of contents and a number of related documents.

1.16 Label Commands

These are commands that can be used within a database. Commands must
start in the very first column of a line. If a line begins with an @
sign, then it is interpreted as a command.

@DATABASE <name>
Must be the very first line of a Amiga HyperText document.

@MASTER <path>
Complete path of the source document used to define this HyperText
database.

@NODE <name> <title>
Indicate the start of a node (page/article/section). The first
node, or main node, must be named MAIN. MAIN must be the master
table of contents for the database.

@DNODE <name>
Indicates the start of a dynamic node. The HyperText system uses the
callback hooks to obtain the document from a document provider.

@INDEX <name/node>
Specify the name of the index node, accessed by the Index button. Can
be a node in an external database.

@REMARK <remark>
Remark (not displayed to the user).

1.17 Node Label Commands

These are commands that can be used within a @NODE.

@ENDNODE <name>
Indicate the end of a node. Must start at the beginning of a line.

@TITLE <title>
Title to display in the title bar of the window during the display of this node. Must start at the beginning of a line.

@TOC <node name>
Name of the node that contains the table of contents for this node. Defaults to MAIN. This is the node that is displayed when the user presses the "Contents" button.

@PREV <node name>
Node to display when the user selects "< Browse"

@NEXT <node name>
Node to display when the user selects "Browse >"

@{<label> <command>}
Indicate a textual link point. Can be anywhere in a line.

1.18 Action Commands

These are commands that can be assigned to a link point.

ALINK <name> <line>
Load the named node into a new window, with <line> at the top of the display.

CLOSE
Close the window (should only be used on windows that where started with alink).

LINK <name> <line>
Load the named node, with <line> at the top of the display.

RX <command>
Execute an ARexx macro.

RXS <command>
Execute an ARexx string file. To display a picture, use 'ADDRESS COMMAND DISPLAY <picture name>', to display a text file 'ADDRESS COMMAND MORE <doc>'.

SYSTEM <command>
Execute an AmigaDOS command.

QUIT

Shutdown the current database.

1.19 Example AmigaGuide Database

The following is an example of an AmigaGuide database. It doesn't contain any 'useful' information, but it does show the usage of some of the commands.

```
@database "example.guide"
@master "example.doc"

@node Main "Example AmigaGuide database"

Table of Contents

   @{"ARexx" link ARexx}
   @{"Shell" link Shell}
   @{"Workbench" link Workbench}

@endnode

@node ARexx
Put something here about ARexx
@endnode

@node Shell
Put something here about the Shell
@endnode

@node Workbench
Put something here about Workbench. Say that it has @{"icons" link icon}.
@endnode

@icon "Workbench Icons"
Those little pictures that you can drag around.
@endnode
```

1.20 AREXX SCRIPTS

It is possible to control AmigaGuide using ARexx. Each occurrence of AmigaGuide has an ARexx port. The AmigaGuide shared system library is also an ARexx function host.

Port Naming

ARexx Commands

ARexx Functions

1.21 Port Naming

The default port name is AMIGAGUIDE.# where # is the occurrence. With the AmigaGuide utility, a port name can be specified as a command line argument. An application with an AmigaGuide interface can also provide the port name.

1.22 ARexx Commands

Any of the following action commands are also ARexx commands. All commands are not case-sensitive.

ALINK <name> <line>
Load the named node into a new window, with <line> at the top of the display.

CLOSE
Close the window (should only be used on windows that where started with alink).

LINK <name> <line>
Load the named node, with <line> at the top of the display.

SYSTEM <command>
Execute an AmigaDOS command.

QUIT
Shutdown the current database.

1.23 ARexx Functions

The amigaguide.library is an ARexx function library. The library can be added as a function host with the following lines:

```
/* Load the HyperText library as a function host */
IF ~SHOW('L','amigaguide.library') THEN
   CALL ADDLIB('amigaguide.library',0,-30)
```

It supports the following functions (function names are not case-sensitive).

ShowNode
PUBSCREEN/K,DATABASE/K,NODE/K,LINE/N
Display a node on the named screen. Defaults to the Main node, on the Workbench screen. If DATABASE isn't specified, then will search through the cross-reference list to get the database name.

LoadXRef

NAME/K
Load a cross-reference file into memory.

GetXRef
NODE/K
Return information on NODE. Format of the text string returned is
"NODE" "DATABASE" TYPE LINE.

ExpungeXRef
,
Flush the cross-reference list from memory.

1.24 ADDING AN AmigaGuide INTERFACE TO YOUR APPLICATION

Applications can add AmigaGuide support, using the functions within
the amigaguide.library.

The HyperApp example on disk illustrates how to add context sensitive
help to an application.

1.25 CROSS REFERENCE FILES

AmigaGuide allows cross-reference tables to be loaded that specify
what document a keyword is defined in. This cross-reference table is
used by the "Find Document" requester to locate a node. It is also
used by the AD2HT utility to construct hypertext versions of the
system Autodoc files.

A cross-reference file follows a layout similar to the devs:mountlist
format. The table itself starts with a line that consists of the
keyword XREF: and ends with a line that contains a # as the only
uncommented character. Comments can be included in the C-style
format, beginning with "/*" and ending with "*/".

```
    /* This is a comment */
    XREF:
  ...
  "Gadget"  "intuition/intuition.h" 215 3
  ...
    #
```

A cross-reference entry consists of four words:

Keyword
The keyword that is being defined.

File
The ASCII file or database that the keyword is defined in.

```
  Line
  The line within the node that the keyword is defined on.

  Type
  This field indicates the type of keyword.  Possible values are.

    0  Generic AmigaGuide link.
    1  Describes a function.
    2  Describes a command.
    3  Points to an include file.
    4  Describes a macro.
    5  Describes a structure.
    6  Describes a structure field.
    7  Describes a type definition.
    8  Describes a define.



                Loading a Cross Reference List

                Access to the Cross Reference List
```

1.26 Loading a Cross Reference List

```
A cross-reference list can be loaded from disk using the LoadXRef()
function.  The format is.


  success = LoadXRef(lock, name);

  LONG success;
  BPTR lock;
  STRPTR name;


The arguments are.


  lock
  Lock on the directory where the file is located.  May be NULL.

  name
  Name of the cross-reference file to load.  LoadXRef will search the
  user preference path.


Returns


  -1  Indicates that the load was aborted by a control-C.

   0  Unable to load the file.
```

```
   1  Successfully loaded the file.

   2  No changes have been made since the last time that this file was
      loaded.
```

1.27 Access to the Cross Reference List

An application can use the GetAmigaGuideAttr() function to obtain a pointer to the cross-reference list. The application then may search through the list, or even save the list to disk. Note that access to this list is read-only, and must be enclosed between a call to LockAmigaGuideBase() and UnlockAmigaGuideBase().

```
    struct List *list;
    LONG key;

    /* Lock the AmigaGuideBase for exclusive access */
    key = LockAmigaGuideBase(NULL);

    /* Get a pointer to the cross-reference list */
    if (GetAmigaGuideAttr(AGA_XRefList, NULL, &list))
    {
  /* Do something with the list */
    }

    /* Unlock AmigaGuideBase */
    UnlockAmigaGuideBase(key);
```

The cross-reference list consist of nodes of struct XRef, defined in <libraries/amigaguide.h>.

```
    /* Cross-reference node */
    struct XRef
    {
        struct Node xr_Node;  /* Embedded node */
        UWORD xr_Pad;   /* Padding */
        struct DocFile *xr_DF;  /* (Private) Document defined in */
        STRPTR xr_File;   /* Name of document file */
        STRPTR xr_Name;   /* Name of item */
        LONG xr_Line;   /* Line defined at */
    };

    #define XRSIZE  (sizeof (struct XRef))
```

Following are the field definitions.

```
  xr_Node
  Embedded node structure.
  xr_Node.ln_Name points to xr_Name.
```

xr_Node.ln_Type contains the type of the keyword.

xr_Pad
Used to align the remaining fields.

xr_DF
Private pointer.

xr_File
Pointer to the name of the file that xr_Name is defined in.

xr_Name
Pointer to the keyword.

xr_Line
The line, within xr_File, that xr_Name is defined on.

1.28 DYNAMIC NODE HOST

AmigaGuide provides a unique feature to hypertext systems, called Dynamic Nodes. A Dynamic Node is a hypertext or plain text document that is generated in real-time as opposed to coming from a static file. An application that generates Dynamic Nodes is called a Dynamic Node Host.

If a link point within a document isn't resolved, it will query a list of Dynamic Node Hosts to see if any one of these external applications can resolve the node.

This feature allows for dynamic interaction with constantly changing data. This feature is useful for AmigaGuide authoring tools, interactive development environments, extremely context sensitive help systems, to name a few.

Dynamic Nodes has been implemented using an Object Oriented Programming paradigm. When a link point hasn't been resolved a HM_FindNode message is sent to each Dynamic Node Host on the list. Once the node has been found, then a HM_OpenNode is sent to the Dynamic Node Host that the node belongs to. HM_CloseNode is sent to the host once the node has been exited.

1.29 Initializing a Dynamic Node Host

In order for an application to register itself as a Dynamic Node Host, it must initialize a hook and add the hook to the AmigaGuide Dynamic Node list, using the AddAmigaGuideHost() library call.

The hook structure as defined in <utility/hooks.h>.

```
    /* Standard hook structure */
    struct Hook
    {
        struct MinNode h_MinNode;
        ULONG (*h_Entry)();   /* assembler entry point */
        ULONG (*h_SubEntry)();    /* often HLL entry point */
        VOID *h_Data;     /* owner specific */
    };
```

The AddAmigaGuideHost() function returns a pointer to an AmigaGuideHost structure. This structure, defined in <libraries/amigaguide.h>, is as follows.

```
    /* Callback handle */
    struct AmigaGuideHost
    {
        struct Hook agh_Dispatcher; /* Dispatcher */
        ULONG agh_Reserved;   /* Must be 0 */
        ULONG agh_Flags;
        ULONG agh_UseCnt;   /* Number of open nodes */
        APTR agh_SystemData;    /* Reserved for system use */
        APTR agh_UserData;    /* Anything you want... */
    };
```

Following are the field definitions for the AmigaGuideHost structure.

agh_Dispatcher
This is a copy of the Hook that was passed to AddAmigaGuideHost().

agh_UserData
Can be manipulated by the Dynamic Node Host any way it sees fit.

The other fields are not to be manipulated in any way.

1.30 Removing a Dynamic Node Host

A Dynamic Node Host is removed using the RemoveAmigaGuideHost() library function. The application must successfully remove the hook before exiting, otherwise AmigaGuide would end up calling the hook function, that has been unloaded from the system, causing a system crash.

The following code fragment illustrates how to initialize and remove
a Dynamic Node Host.

```
    #include <exec/types.h>
    #include <libraries/amigaguide.h>
    #include <clib/exec_protos.h>
    #include <clib/amigaguide_protos.h>
    #include <pragmas/exec_pragmas.h>
    #include <pragmas/amigaguide_pragmas.h>
    #include <stdio.h>

    extern struct Library *SysBase, *DOSBase;
    struct Library *AmigaGuideBase;

    #define ASM __asm
    #define REG(x)  register __ ## x

    ULONG __saveds dispatchDNH(struct Hook *, STRPTR, Msg);
    ULONG ASM hookEntry(REG(a0) struct Hook *,REG(a2) VOID *,REG(a1) VOID *);

    /* Callback hook dispatcher */
    ULONG __asm hookEntry (
    REG(a0) struct Hook *h,
    REG(a2) VOID *obj,
    REG(a1) VOID *msg)
    {
        /* Pass the parameters on the stack */
        return ((h->h_SubEntry)(h, obj, msg));
    }

    main (int argc, char **argv)
    {
  struct Hook hook;
  AMIGAGUIDEHOST hh;

  /* amigaguide.library works with 1.3 and newer versions of the OS */
  if (AmigaGuideBase = OpenLibrary ("amigaguide.library", 33))
  {
      /* Initialize the hook */
      hook.h_Entry = hookEntry;
      hook.h_SubEntry = dispatchDNH;

      /* Add the AmigaGuideHost to the system */
      if (hh = AddAmigaGuideHost (&hook, "ExampleHost", NULL))
      {
    printf ("Added AmigaGuideHost 0x%lx\n", hh);

    /* Wait until we're told to quit */
    Wait (SIGBREAKF_CTRL_C);

    printf ("Remove AmigaGuideHost 0x%lx", hh);

    /* Try removing the host */
    while (RemoveAmigaGuideHost (hh, NULL) > 0)
    {
        /* Wait a while */
```

```
        printf (".");
        Delay (250);
    }
    printf ("\n");
      }
      else
      {
    printf ("Couldn't add AmigaGuideHost\n");
      }

      /* close the library */
      CloseLibrary (AmigaGuideBase);
  }
    }
```

1.31 Handling Dynamic Node Host Messages

Once the Dynamic Node Host has been added to AmigaGuide, it can start receiving messages for different requests.

Currently, AmigaGuide supports the following methods, or message types, for a Dynamic Node Host.

HM_FindNode
When AmigaGuide can't resolve a link, then it sends a HM_FindNode message to all Dynamic Node Hosts to see which host defines the node.

HM_OpenNode
Once AmigaGuide locates the host that defines a node, using the HM_FindNode message, then the HM_OpenNode message is sent to that host to ask it to open the node.

HM_CloseNode
Once the user has closed all occurrences of a Dynamic Node, then AmigaGuide sends the HM_CloseNode message to the host that opened the node.

HM_Expunge
AmigaGuide sends this message to all Dynamic Node Hosts when the Expunge vector of amigaguide.library is invoked, or the ExpungeDataBases() function is called.

Several of the methods receive a TagItem array as an argument. Currently the following tags are supported. The tag values are defined in <libraries/amigaguide.h>.

HTNA_Screen
A pointer to the screen on which source AmigaGuide window resides.

HTNA_Pens
The pen array associated with the screen.

HTNA_Rectangle
A Rectangle structure (defined in <graphics/gfx.h>) containing the dimensions of the window.

Each method requires one or more parameters. The MethodID is the only common parameter for each method.

1.32 HM_FindNode

Used to locate the Dynamic Node Host that a node is defined by. When a Dynamic Node Host receives a HM_FindNode message for a node that it owns, it should reply with TRUE, otherwise it must respond with FALSE.

The HM_FindNode method receives the following arguments:

```
/* HM_FindNode */
struct opFindHost
{
    ULONG MethodID;
    struct TagItem *ofh_Attrs;  /*  R: Additional attributes */
    STRPTR ofh_Node;    /*  R: Name of node */
    STRPTR ofh_TOC;     /*  W: Table of Contents */
    STRPTR ofh_Title;   /*  W: Title to give to the node */
    STRPTR ofh_Next;    /*  W: Next node to browse to */
    STRPTR ofh_Prev;    /*  W: Previous node to browse to */
};
```

The field definitions are as follows

ofh_Attrs
This field contains a pointer to a TagItem array of attributes for the message. This field is read-only.

ofh_Node
The name of the node to open. This field is read-only. It is possible for this name to contain parameters that need to be parsed. For example, the command that triggered the link could have been:

```
Link "snd/beep 320"
```

In which case, the ofh_Node field would contain:

```
    beep 320
```

ofh_TOC
The Table of Contents to assign to this node. This is the name of the node to link to, if the "Contents" button is pressed. This field can be written to (not implemented).

ofh_Title
The title to assigned to this node. This field can be written to.

ofh_Next
The name of the logical next node. This is the name of the node to link to if the "Browse >" button is pressed. This field can be written to.

ofh_Prev
The name of the logical previous node. This is the name of the node to link to if the "< Browse" button is pressed. This field can be written to.

1.33 HM_OpenNode

Once AmigaGuide locates the host that defines a node, using the HM_FindNode message, then the HM_OpenNode message is sent to that host to ask it to open the node. If the Dynamic Node Host is able to open the node, then it should respond with TRUE, otherwise respond with FALSE.

The HM_OpenNode method receives the following arguments:

```
    /* HM_OpenNode, HM_CloseNode */
    struct opNodeIO
    {
        ULONG MethodID;
        struct TagItem *onm_Attrs;  /*  R: Additional attributes */
        STRPTR onm_Node;    /*  R: Node name and arguments */
        STRPTR onm_FileName;    /*  W: File name buffer */
        STRPTR onm_DocBuffer;   /*  W: Node buffer */
        ULONG onm_BuffLen;    /*  W: Size of buffer */
        ULONG onm_Flags;    /* RW: Control flags */
    };
```

The field definitions are as follows

onm_Attrs
This field contains a pointer to a TagItem array of attributes for the message. This field is read-only.

onm_Node
The name of the node to open. This field is read-only. It is possible for this name to contain parameters that need to be parsed. For example, the command that triggered the link could have been:

```
    Link "snd/beep 320"

  In which case, the onm_Node field would contain:

    beep 320

  onm_FileName
  If you want AmigaGuide to read a particular node from disk, then supply
  the file name here.  The file can either be a straight ASCII file or an
  AmigaGuide document (not a database).  The application can write to
  this field.

  onm_DocBuffer
  If you are dynamically creating a node in memory, then use this field
  to point to the buffer.  If this field is used, then the onm_BuffLen
  field must be filled in also.  The application is in charge of freeing
  onm_DocBuffer when it is done (indicated by a HM_CloseNode message).
  The application can write to this field.

  onm_BuffLen
  The length of the buffer that onm_DocBuffer points to.  The
  application can write to this field.

  onm_Flags
  These are control flags that the Dynamic Node Host can set.

    HTNF_KEEP - Don't flush this node from memory until the database is
    closed.  This will delay the HM_CloseNode message until the database
    is closed.

    HTNF_ASCII - The node is straight ASCII, doesn't contain any
    AmigaGuide keywords.

    HTNF_CLEAN - Remove the node from the database as soon as it is
    closed.

    HTNF_DONE - This flag is used to indicate to AmigaGuide that the
    Dynamic Node Host already took care of presenting the node, and
    that there is no need for AmigaGuide to present it.  This is
    useful for playing sounds, animations, or even debugging
    information.
```

1.34 HM_CloseNode

```
Once the user has closed all occurrences of a Dynamic Node, then
AmigaGuide sends the HM_CloseNode message to the host that opened the
node.  If the Dynamic Node Host is able to close the node, then
respond with TRUE, otherwise respond with FALSE.

The HM_CloseNode message uses the same message structure as HM_OpenNode.
```

1.35 HM_Expunge

AmigaGuide sends this message to all Dynamic Node Hosts when the Expunge vector of amigaguide.library is invoked, or the ExpungeDataBases() function is called. The Dynamic Node Host should free as much memory as it possibly can.

The HM_Expunge method receives the following arguments:

```
/* HM_Expunge */
struct opExpungeNode
{
    ULONG MethodID;
    struct TagItem *oen_Attrs;  /*  R: Additional attributes */
};
```

The field definitions are as follows

oen_Attrs
Currently, no attributes passed.

1.36 Message Dispatcher

The following is an example of Dynamic Node Host message dispatcher. Since this is executed with a callback hook, it is being run on the calling process' task, not the Dynamic Node Host process. Because of that, it is necessary to load the global data segment using geta4() or __saveds.

```
ULONG __saveds
dispatchAmigaGuideHost (struct Hook *h, STRPTR db, Msg msg)
{
    struct opNodeIO *onm = (struct opNodeIO *) msg;
    ULONG retval = 0;

    switch (msg->MethodID)
    {
        /* Does this node belong to you? */
        case HM_FindNode:
            {
                struct opFindHost *ofh = (struct opFindHost *) msg;

                DB (kprintf ("Find [%s] in %s\n", ofh->ofh_Node, db));

                /* See if they want to find our table of contents */
                if ((stricmp (ofh->ofh_Node, "main")) == 0)
                {
                    /* Return TRUE to indicate that it's your node,
                     * otherwise return FALSE. */
                    retval = TRUE;
```

```
            }
            else
            {
                /* Display the name of the node */
                Display (onm);

                /* Return TRUE to indicate that it's your node,
                 * otherwise return FALSE. */
                retval = FALSE;
            }
        }
        break;

    /* Open a node. */
    case HM_OpenNode:
        DB (kprintf ("Open [%s] in %s\n", onm->onm_Node, db));

        /* See if they want to display our table of contents */
        if ((stricmp (onm->onm_Node, "main")) == 0)
        {
            /* Provide the contents of the node */
            onm->onm_DocBuffer = TEMP_NODE;
            onm->onm_BuffLen = strlen (TEMP_NODE);
        }
        else
        {
            /* Display the name of the node */
            Display (onm);

            /* Indicate that we want the node removed from our
             * database, and that we handled the display of the
* node */
            onm->onm_Flags |= (HTNF_CLEAN | HTNF_DONE);
        }

        /* Indicate that we were able to open the node */
        retval = TRUE;
        break;

    /* Close a node that has no users. */
    case HM_CloseNode:
        DB (kprintf ("Close [%s] in %s\n", onm->onm_Node, db));

        /* Indicate that we were able to close the node */
        retval = TRUE;
        break;

    /* Free any extra memory */
    case HM_Expunge:
        DB (kprintf ("Expunge [%s]\n", db));
        break;

    default:
        DB (kprintf ("Unknown method %ld\n", msg->MethodID));
        break;
}
```

```
        return (retval);
    }
```

1.37 DEVELOPER SPECIFIC UTILITIES

On the DevCon disks are a number of utilities for AmigaGuide that would be of special interest to the developer.

AD2HT
Scans Autodocs for function and command names, scans the INCLUDE: directory for .h include files, and scans the include files for structure definitions. Also parses Autodoc files and constructs a corresponding AmigaGuide database. It resolves links to functions, commands, include files and structures.

MKProto
Scans a source directory and builds a cross-reference showing where all functions and structures are defined. Something like ctags on the Sun.

1.38 GLOSSARY

Autodoc
Documentation extracted from source code.

browse
Navigate sequentially through a series of documents, instead of via links.

cross-reference table
A table that consists of the following information:

keyword
A word or phrase

database
Name of the database that the keyword is defined in.

line
The line that the keyword is defined on within the database. This only applies if the database is a straight text file (such as an include file).

type
What type the keyword is, such as a 'normal', function, command, include file, or structure.

database
A file that consists of multiple documents.

document
A block of text, constrained to one subject. Also called a node.

Dynamic Node
  A Dynamic Node is a hypertext, or plain text, document that is
  generated in real-time, or from live data, as opposed to coming from a
  static file.

Dynamic Node Host
  An application that generates Dynamic Nodes.

link
  A word, or phrase, within a document that is linked to another
  document.

node
  A block of text, constrained to one subject. Also called a document.

retrace
  To follow, in a reverse direction, the path taken through a series of
  documents.

table of contents
  A list of documents, categorized by type.

1.39 RECOMMENDED READING

"Hypertext Hands-On!" Ben Shneiderman & Greg Kearsley. Addison-Wesley
Publishing Company. ISBN 0-201-13546-9

"Understanding Hypertext Concepts and Applications" Philip Seyer.
Windcrest Books. ISBN 0-8306-9108-1 (hard) 0-8306-3308-1 (pbk.)