HiTest

Enhanced Lotus Notes API

Version 1.0

Programmer's Reference

Edge Research Inc.

Edge Research Inc.
One Harbour Place, Suite 455
Portsmouth, NH 03801
(USA)

Phone: (603) 431-5300
FAX: (603) 427-2541

TABLE OF CONTENTS

1. OVERVIEW OF HITEST....1

1.1 General....1
1.2 Benefits....1
1.3 Basic Structure....2

2. INSTALLING HITEST....3

2.1 Installation....3
2.2 Samples....4

3. PROGRAMMING TO THE HITEST API....5

3.1 Requirements....5
3.2 Program Flow....6
3.3 Data Types....10
3.4 Context....12
3.5 Error Handling....15
3.6 Mixing HiTest with the standard Notes API....16

4. HITEST FUNCTIONS....18

4.1 Overview....18
4.2 Function Descriptions....20
(Global)....21
htConvert....22
htConvertLength....24
htGetEnvString....25
htGetInfo....26
htInit....27
htSetEnvString....28
htSetOption....29
htTerm....31
Addin....32
htAddinGetInterval....33
htAddinPutMsg....34
htAddinSetInterval....35
htAddinSetStatus....36
htAddinYield....37
Cell....38
htCellBind....39
htCellFetch....41
htCellLength....42
htCellUnbind....43
Column....44
htColumnCount....45
htColumnList....46
Composite....47
htCompCopy....48
htCompCopySubset....49
htCompCreate....51
htCompExport....52
htCompExportList....53
htCompGetInfo....54

htCompGetOSFont....55
htCompImport....56
htCompImportList....57
htCompListText....58
htCompMerge....60
htCompPutOSFont....61
Comprec....62
htComprecCount....67
htComprecDelete....68
htComprecFetch....69
htComprecGetPtr....71
htComprecInsert....73
htComprecLength....74
htComprecList....75
htComprecUpdate....76
Cursor....77
htCurClose....78
htCurGetInfo....79
htCurOpen....81
htCurReset....82
htCurSetOption....83
Database....84
htDbGetPath....85
htDbList....86
htDbListCat....88
Datetime....90
htDatetimeCompare....91
htDatetimeCreate....92
htDatetimeDiff....93
htDatetimeGetInfo....94
htDatetimeUpdate....95
Document....96
htDocClose....97
htDocCopy....98
htDocCreate....99
htDocDelete....100
htDocFetch....101
htDocGetInfo....102
htDocOpen....104
htDocPut....105
htDocUpdate....106
Error....107
htErrorFetch....108
htErrorSetBuffer....109
htErrorSetProc....110
Field....111
htFieldCount....112
htFieldGetInfo....113
htFieldList....115
File....116
htFileDelete....117
htFileFetch....118
htFileList....119
htFilePut....120
Font....121
Form....122
htFormCopy....124
htFormDelete....125
htFormGetAttrib....126
htFormGetId....127
htFormList....128

htFormSet....129
htFormTemplate....130
Formula....131
htFormulaConcat....132
htFormulaConcatf....133
htFormulaCopy....134
htFormulaExec....135
htFormulaLength....136
htFormulaReset....137
Index....138
htIndexCount....139
htIndexGetInfo....140
htIndexGetPos....141
htIndexGetTreePos....142
htIndexNavigate....143
htIndexRefresh....145
htIndexSearch....146
htIndexSetPos....148
htIndexSetTreePos....149
Item....150
htItemBind....152
htItemCount....154
htItemDelete....155
htItemFetch....156
htItemGetInfo....158
htItemGetPtr....159
htItemLength....161
htItemList....162
htItemPut....163
htItemUnbind....164
Macro....165
htMacroCopy....166
htMacroDelete....167
htMacroExec....168
htMacroGetId....170
htMacroList....171
Mail....172
htMailSend....173
Server....175
htServerExec....176
htServerGetInfo....177
htServerList....178
TextList....179
htTextListCount....180
htTextListFetch....181
htTextListGetPtr....182
htTextListLength....183
View....184
htViewCopy....186
htViewDelete....187
htViewGetAttrib....188
htViewGetId....189
htViewList....190
htViewSet....191

GLOSSARY....192

INDEX....197

1. Overview of HiTest

1.1 General

The HiTest API (Application Programming Interface) is an alternative higher level C interface to the API provided by Lotus for Lotus Notes. Program development is significantly faster and requires a fraction of the API code needed with the standard API.

HiTest API programmers should have some familiarity with C programming. Also useful is some familiarity with either Lotus Notes or the standard Lotus Notes API. The glossary defines terms related to Lotus Notes and the HiTest API.

This manual is organized into the following chapters:

1. Overview of the HiTest API
2. Instructions and requirements for installing HiTest
3. Instructions for building HiTest API applications
4. HiTest function descriptions

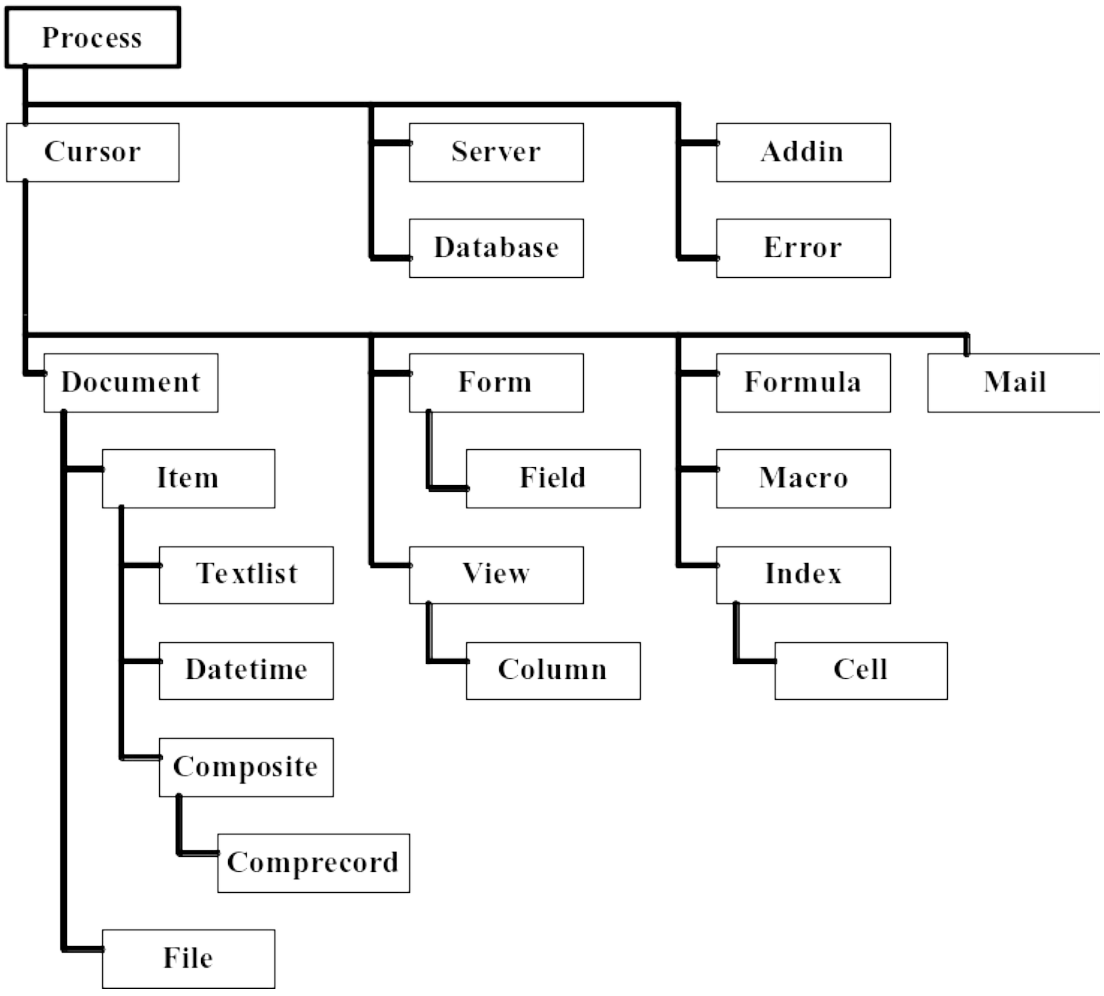
1.2 Benefits

Some of the major benefits of the HiTest API are:

- Protective API layer catches invalid handles and operations which, under the standard API, cause crashes.
- Simple browsing functions for everything from servers and databases through attachments and items.
- Abstraction of metadata (forms and views) for quick, single function access to form and view information. Additional field and column browsing functions let developers avoid completely the internal Notes BLOBs containing the metadata.
- Simple data transfer -- by binding document items and view cells, a single function call will load, store, or update multiple items and cells.
- Automatic data conversion implicitly when fetching or storing data, or explicitly with a single conversion function.
- Internal handling of components that are cumbersome in the standard Notes API, such as ID tables, collections, memory blocks, and composite data.
- Simple high-level creation, access, and manipulation of composite data (including single-function import and export) far above the standard API's byte level with composite data functions.
- Integrated single-function support for macro execution and full text search.
- Support for server or client add-in programs with a built-in scheduler.
- Remote server console support lets clients control server activities such as replication and program execution.
- Automatic management of response hierarchies, including the ability to copy or delete an entire hierarchy of unlimited breadth and depth with a single function call, as well as easy creation and management of responses.
- Advanced mail interface for sending mail from a structure, from an existing document, from bound memory, or from any combination with a single function call.
- Three error handling methods so developers can use the method with which they are most comfortable.
- High-level object-based HiTest API facilitates rapid program development.

1.3 Basic Structure

HiTest offers a consistent interface to the various components of Lotus Notes. The highest level of abstraction is the set of HiTest objects, covering the various components of Lotus Notes. The following diagram shows the object containment hierarchy:



The object-based design of HiTest has two major usability benefits. First, abstraction of objects that the standard API simply considers data (form, field, view, column, macro, cell, file, and composite) often lets single function calls replace medium-sized functions. Second, HiTest was designed as a consistent high-level Notes interface, rather than a collection of low-level internal functions. Common actions, such as iterating through an object, are done through common interfaces (e.g., most objects contain a List function, which works similarly across objects).

2. Installing HiTest

2.1 Installation

The HiTest API is built over the standard Notes API. Therefore, any machine used to build or run programs with HiTest must have a licensed copy of the Lotus Notes software (client or server, version 3.0 or higher). In addition, the machine must satisfy the hardware and software requirements to run the Lotus Notes software.

The HiTest API consists of the following files:

HTNOTES.H	Include in all programs calling HiTest
HTxNOTES.LIB	Import library for the HiTest DLL
HTxNOTES.DLL	HiTest DLL - required to run HiTest programs

The import library and DLL filenames contain an 'x', which varies with the operating system. The ‘x’ is replaced with 'W' for Windows 3.1, 'O' for OS/2 1.3 (16-bit), and ‘2’ for OS/2 2.x (32-bit).

The HiTest API is installed from a ZIP file. First, create a HiTest API directory (e.g., C:\HITEST) and copy the file into this directory. Next, unzip the ZIP file from either a DOS window or an OS/2 window with the HiTest directory as the current directory. Use the *-d* option when unzipping to extract the subdirectories as well as the files. The installation process creates the following directories and files under the HiTest installation directory:

INCLUDE directory	
HTNOTES.H	HiTest API include file
LIB directory	
HTWNOTES.LIB	Windows 3.1 import library
HTONOTES.LIB	OS/2 1.3 16-bit import library
HT2NOTES.LIB	OS/2 2.x 32-bit import library
DLL directory	
HTWNOTES.DLL	Windows 3.1 DLL
HTONOTES.DLL	OS/2 1.3 16-bit DLL
HT2NOTES.DLL	OS/2 2.x 32-bit DLL
SAMPLES directory	Sample programs in subdirectories
...	Each subdirectory contains one sample program
DOC directory	
HITEST.RTF	Microsoft RTF format HiTest reference manual
HITEST*xx*.NSF	Notes database format HiTest reference manual *xx* represents the version number (e.g., HITEST10.NSF for HiTest v1.0)

Next, modify the INCLUDE and LIB environment variables needed by the compiler to find the HiTest files. Add the HiTest INCLUDE directory to the INCLUDE environment variable for the compiler to find the HTNOTES.H include file. Add the HiTest LIB directory to the LIB environment variable for the linker to find the proper import library. To run HiTest API programs, the HiTest DLL must be available to the operating system. For Windows, the DLL is usually placed in the WINDOWS\SYSTEM directory, but may be in the WINDOWS directory or elsewhere in the PATH. For OS/2, the DLL should be in a directory in the LIBPATH environment variable.

To run HiTest API programs, the NOTES.INI file must be available in a directory in the PATH environment variable. Most Notes installations locate this file in either the Windows or Notes program directory, in which case it should

already be in the PATH. Some installations, though, locate this file in the Notes data directory. If this is the case, then add this directory (or another directory if NOTES.INI is elsewhere) to the PATH if not already included.

2.2 Samples

The HiTest installation includes various sample programs. These programs demonstrate common activities performed with the HiTest API. One subdirectory under the SAMPLES directory exists for each sample program. Each sample program includes the following files:

README.TXT	describes the sample program
MAKEFILE	builds the sample program. Invoke with an operating system constant (e.g.: “NMAKE WIN”, “NMAKE OS21”, or “NMAKE OS22”).
HTSAMPLE.H	generic sample program header file
PROGRAM.C	sample program-specific source code
PROGRAM.H	optional sample program header file

Build and run the sample program SIMPLE to test for proper installation of HiTest. The other sample programs are described below:

Program	Description
DBMETA	Demonstrates database and metadata access by connecting to a database and generating all metadata for the database to a file.
VIEWDMP	Demonstrates creation and navigation of a view-based index and its cell data. The view is rendered without formatting to a file.
DOCITEM	Demonstrates document item access by item binding and by direct item access. The item data for all documents in a flat index is written to a file.
DOC2NSF	Converts and parses the HiTest documentation into a Notes database. The documentation is imported into a composite item, which is then parsed to produce one Notes document for each function in the HiTest API. This sample is a detailed example of creation and manipulation of composite data.

3. Programming to the HiTest API

3.1 Requirements

This section contains requirements and instructions for writing HiTest API programs.

The different HiTest platforms support different compilers. Building Windows 3.1 programs requires the Microsoft C compiler version 7.0 or greater. Building OS/2 1.3 16-bit programs requires the Microsoft C compiler version 6.0 (the last version which supported OS/2). Building OS/2 2.x 32-bit programs requires the IBM C Set++ compiler, as well as certain compiler options to interact properly with the 16-bit Notes code beneath HiTest. The options are /Gt (tiled memory to avoid crossing 64K boundaries) and /Sp1 (1-byte structure packing). The /Gs option (remove stack probes) is not required, but is recommended. Programs which call the HiTest API must be compiled as large memory model programs (on 16-bit platforms where memory model is relevant). The minimum stack size for HiTest API programs is 10K (15K for OS/2 2.x), and programs of moderate size or complexity should increase the stack size beyond 10K.

The following table summarizes requirements by platform:

Platform	Requirements
Windows 3.1	Microsoft C compiler version 7.0 or greater large memory model (compiler option /AL) 10K or greater stack OS_WIN compilation constant
OS/2 1.3 (16-bit)	Microsoft C compiler version 6.0 large memory model (compiler option /AL) 10K or greater stack OS_OS21 compilation constant
OS/2 2.x (32-bit)	IBM C Set++ compiler 15K or greater stack OS_OS22 compilation constant Compiler options /Gt and /Sp1 (The use of compiler option /Gs is recommended)

When building a program, to set the stack size in a module definition file use the following statement:

```
STACKSIZE 10240
```

To set the stack size on the linker command line use the following link option:

```
/ST:10240
```

Increase the value for larger programs.

All source code modules using HiTest must declare an operating system before including the HTNOTES.H include file. Define one of the OS constants OS_WIN, OS_OS21, or OS_OS22. To define the constant from within a C program, use the following statement:

```
#define OS_WIN
```

To define the constant during compilation, use the following compilation option:

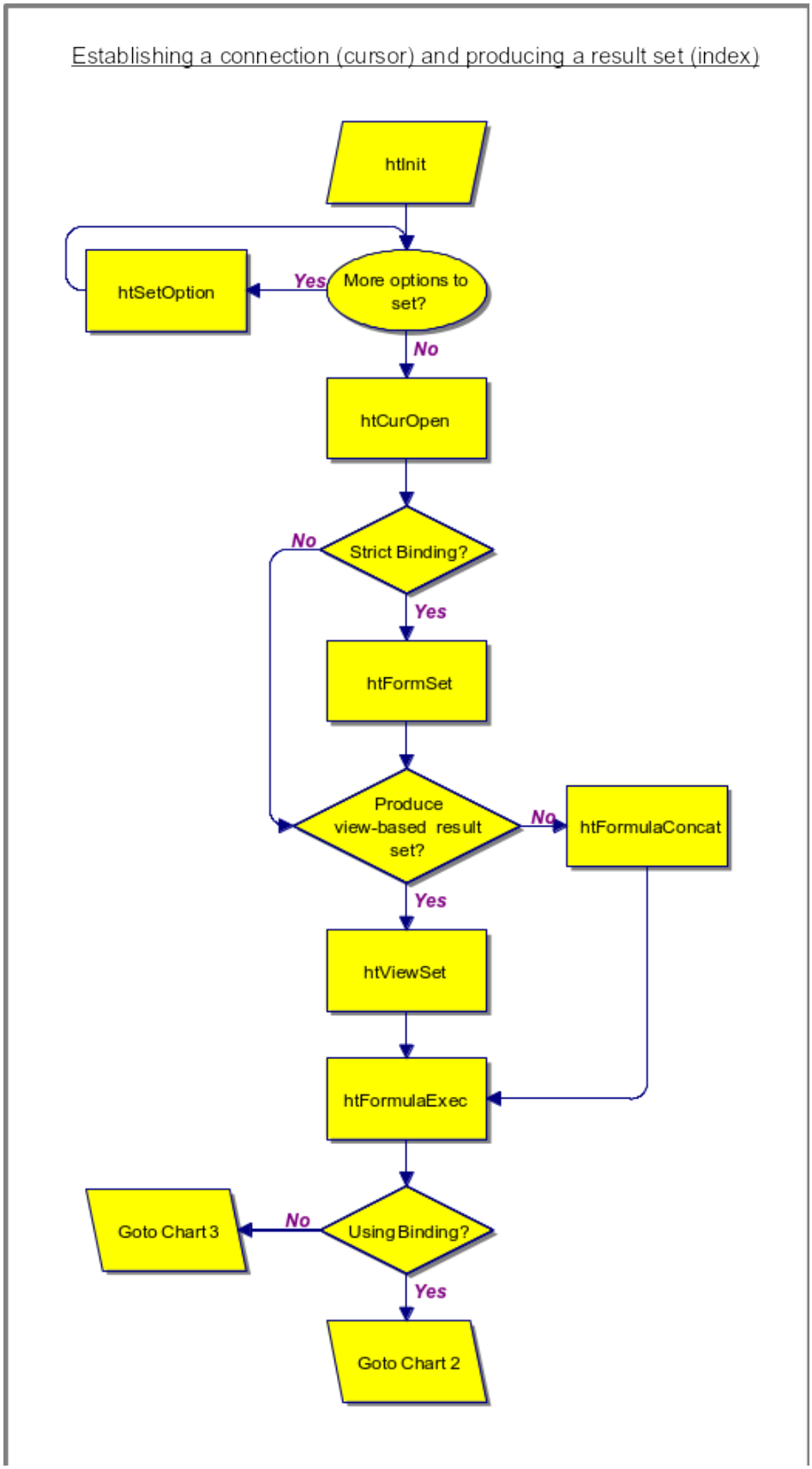
```
/DOS_WIN
```

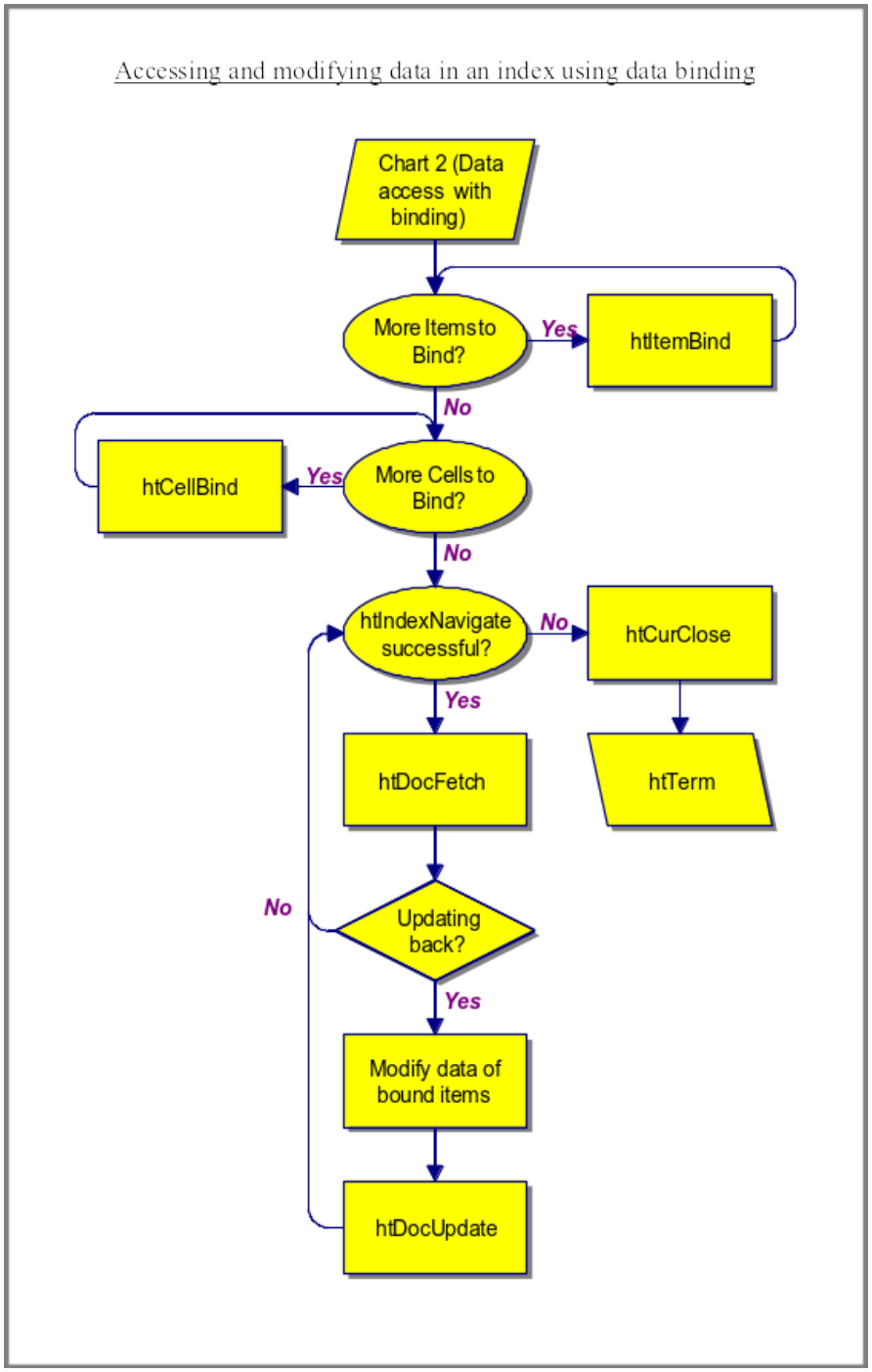
The compilation option supports multiple compilations of the same program with different OS constants to run under multiple operating systems.

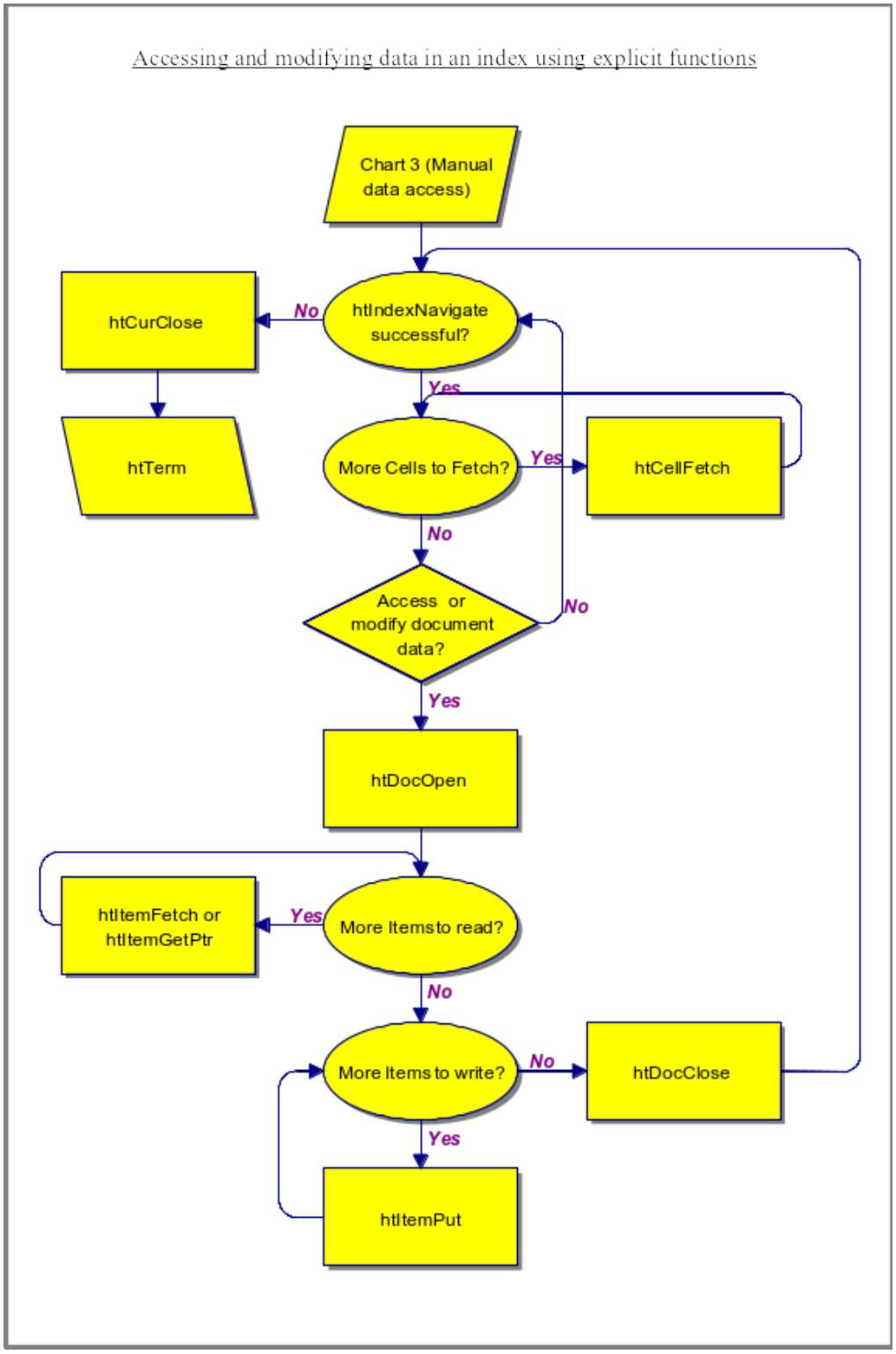
Every HiTest API program must initialize and terminate the HiTest API. All other HiTest functions will fail unless preceded by a call to the initialization function htInit. Additionally, the program must call the htTerm function after all HiTest function calls are complete and before the program terminates. Calling the termination function is crucial to avoid leaving the system in a dangerous state.

3.2 Program Flow

A basic HiTest API program involves reading and writing Notes data. The flowcharts on the following pages show basic program flow including opening a cursor, producing an index (result set), reading and writing data, and closing the connection.


Establishing a connection (cursor) and producing a result set (index)
htInit
More options to set?
Yes
htSetOption
No
htCurOpen
Strict Binding?
No
Yes
htFormSet
Produce view-based result set?
No
htFormulaConcat
Yes
htViewSet
htFormulaExec
Using Binding?
No
Goto Chart 3
Yes
Goto Chart 2


Accessing and modifying data in an index using data binding
Chart 2 (Data access with binding)
More Items to Bind?
Yes
htItemBind
No
htCellBind
Yes
More Cells to Bind?
No
htIndexNavigate successful?
No
htCurClose
Yes
htDocFetch
htTerm
No
Updating back?
Yes
Modify data of bound items
htDocUpdate


Accessing and modifying data in an index using explicit functions
Chart 3 (Manual data access)
htIndexNavigate successful?
No
htCurClose
htTerm
Yes
More Cells to Fetch?
Yes
htCellFetch
No
Access or modify document data?
No
Yes
htDocOpen
htItemFetch or htItemGetPtr
Yes
More Items to read?
No
More Items to write?
No
htDocClose
Yes
htItemPut

3.3 Data Types

This section describes the HiTest API data types. Standard Notes data comes in three 'flavors':

5. Single value data types (NUMBER, TEXT, and DATETIME) contain a single data value (e.g., 3.14).
6. Multiple value data types (TEXT LIST, NUMBER LIST, and TIME LIST) contain one or more values of similar types (e.g., "ABC", "DEF" is a two-element text list). For number or time lists, each individual value is either an exact value or a two-value range (e.g., 3, 5, 7-10 is a three-element number list, where the third element is a range).
7. Composite data type (COMPOSITE) is a collection of individual composite records of different subtypes. Composite data may contain multiple record types, including formatted text, images, links, etc. For a full description of composite record types, see the composite record section in the function descriptions.

HiTest supports all the standard Notes data types. In addition, HiTest supports a special data type (REF) which is a response's reference to a parent document. The constant HTLEN_ITEM_DATA defines the maximum data length for all types except composite. Composite data may be of unlimited length.

The following table describes the data types, with internal formats and any notes. The constant representing each datatype is constructed by prefixing HTTYPE_ (e.g., HTTYPE_INT).

Standard types	Description
INT	C long integer. Although Notes stores all numbers as double-precision floating-point values, many numeric items represent integer values. In these cases it is convenient to use the integer type, which HiTest automatically converts to and from the Notes internal number format.
NUMBER	C double precision floating-point. This is the type used by Notes to store all numeric values. The constant HTLEN_NUMBER_TEXT defines the maximum length of a NUMBER converted to text.
TEXT	Variable length text string. Although Notes stores text without null terminators, all text values transferred to and from the HiTest API use a NULL terminator. It is crucial that all text buffers allow an extra byte for the NULL terminator. The terminator is not included in the text length (e.g., the length of "ABCDE" is five bytes, but a buffer supplied by the calling program must be six bytes). This is true not only for data, but for informational items (e.g., server names, error messages, etc.). The exception is functions that retrieve direct data pointers (GetPtr functions), where text values may not contain a NULL terminator. Handle these values by length rather than with standard C string functions.
DATETIME	8-byte data object representing a date and time. Use the htDatetime functions to manipulate datetime values (do not directly manipulate the datetime data). The HiTest representation is the same as the standard Notes API representation. The range is from year 1 through year 32767, and the precision is one-hundredth of a second. Datetime values are time zone specific. The constant HTLEN_DATETIME_TEXT defines the maximum length of a DATETIME converted to TEXT.

TEXT_LIST	A multiple-value collection of zero or more text strings. Use the htTextList functions for easy access to the contents of a text list. The text values within a text list are not NULL terminated. The format of a text list containing N entries is: WORD number of entries (N) WORD length of entry #1 WORD length of entry #2 ... WORD length of entry #N text text of entry #1 text text of entry #2 ... text text of entry #N
NUMBER_LIST	A multiple-value collection of zero or more C double-precision floating-point values and ranges. The format of a number list containing N values and M ranges is: WORD number of values (N) WORD number of ranges (M) double value #1 double value #2 ... double value #N double low value of range #1 double high value of range #1 ... double low value of range #M double high value of range #M
TIME_LIST	A multiple-value collection of zero or more DATETIME values and ranges. The format of a time list is the same as a number list, with HTDATETIME values in place of doubles.
COMPOSITE	A composite is an ordered list of composite records. Each composite record is of a type within the HTCOMP_xxx enumeration. Unlike other Notes types stored as a single document item, a composite may consist of multiple items. One of the benefits of the HiTest API is the abstraction of a composite object as a single object regardless of the number of document items in that composite. For more information, see the htComp and htComprec function descriptions.

Nonstandard types	**Description**
REF	The HiTest representation of a reference to a parent document. Notes stores response information within the child as a reference to the parent. Reference items are always of the same item name -- "$REF" (use the constant HTNAME_REF). In HiTest, a reference item's value is the HTDOCID of the parent document (internally, Notes uses a more complex representation). Forms do not contain reference fields, so reference items are handled independently of strict binding and form fields.

HiTest supports implicit and explicit data conversion between types. Use the htConvert function to perform explicit conversion. Implicit conversion occurs in functions which fetch or put data (htDocFetch, htDocPut, htDocUpdate, htItemFetch, htItemGetPtr, htItemPut, htCellFetch). While HiTest supports most conversions, certain conversions do not make sense and are invalid. No conversions between numeric (INT, NUMBER, and NUMBER LIST) and time

(DATETIME and TIME LIST) are allowed. Additionally, COMPOSITE can only be converted to and from TEXT. REF cannot be converted. If an unsupported conversion is required, perform two conversions, using type TEXT as the intermediary (e.g., convert from type X to TEXT, and then from TEXT to type Y). Certain conversions may result in a loss of information (e.g., converting a composite item to text will retain only ASCII text information).

3.4 Context

Objects within the HiTest API are represented by either an identifier or a handle. Simple objects need only an identifier. An identifier's type depends on the object. For example, a server's identifier is a string, a document's identifier is an ID, and a view column's identifier is a number. Larger or more complex objects must be 'opened' and 'closed' and require a handle. Cursors, documents, and composite items use handles. Certain actions on objects are only valid within the context of other objects. The simplest example is that of an open document. Closing a cursor containing an open document closes the document and invalidates its handle.

A cursor represents a single HiTest session. Each process or task can contain multiple cursors at any point in time. Multiple cursors may be open to a single database. All actions performed against a database occur through a cursor. A cursor contains a state which consists of an active form, an active view, an index, bindings, and open documents. Any operation which cancels or replaces part of a cursor's state destroys the previous value. For example, producing a new index destroys the previous index.

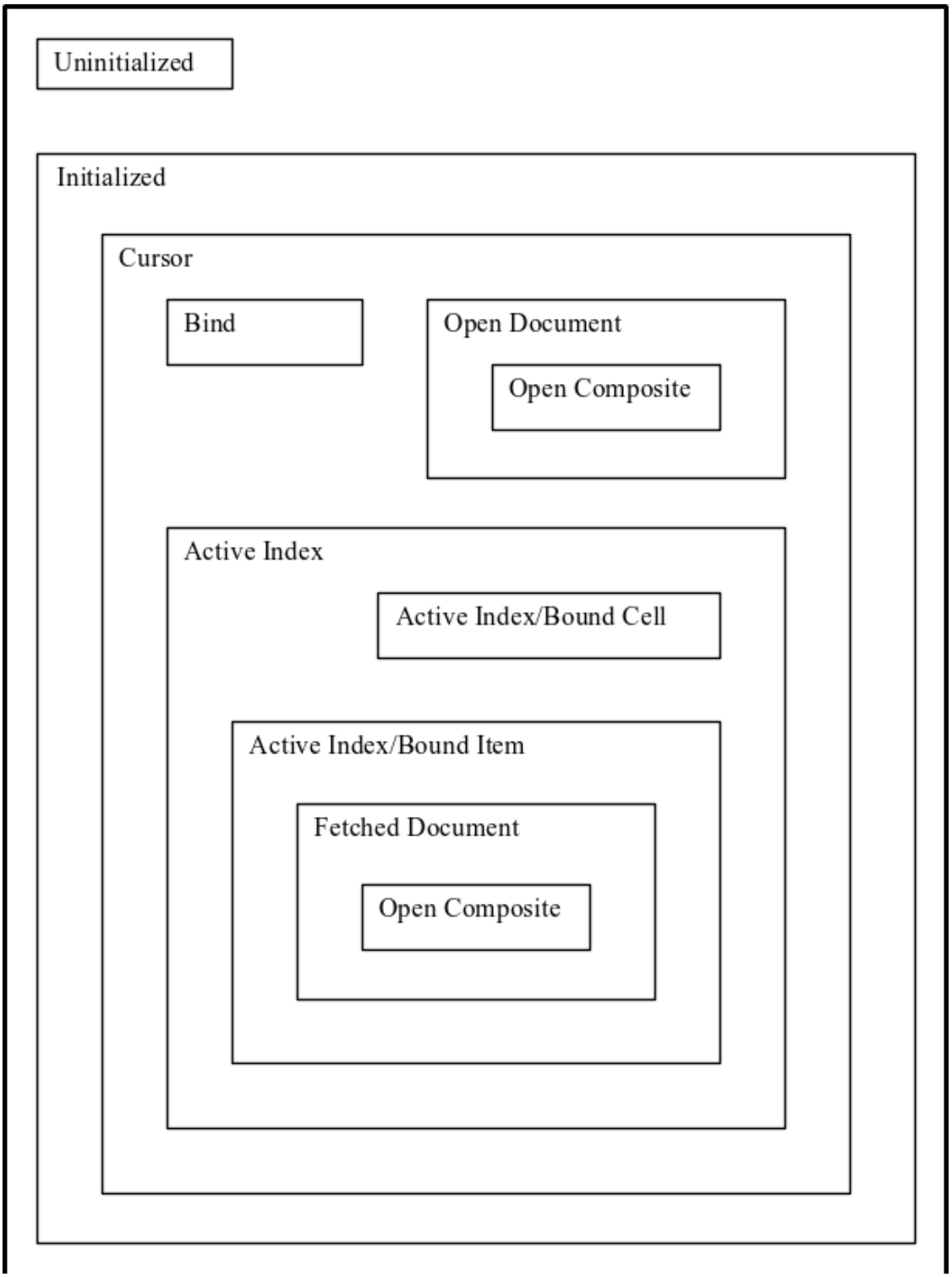
The following table lists the HiTest functional contexts in a simplified state transition table. Each context lists objects and functions which are valid from within that context (an object name indicates that all functions of that object are valid in the context). The context ordering is from highest level to lowest. When entering a lower context from a higher context, all functions in the previous context are still valid . For example, after entering the *Open Composite* context from the *Open Document* context, the htItemPut function is still valid even though it is not listed in the *Open Composite*. Each context also lists those functions which cause a transition to a different context. When entering a lower context from a higher context, all transitions in the previous context are still valid. For example, performing htCurClose from the *Open Document* context causes a return to the *Initialized* context. Multiple contexts may exist simultaneously. For example, calling htDocCreate when in the *Fetched Document* context will create a valid *Open Document* context. These document contexts may operate independently, but a call to htCurClose would close them both by closing their 'parent' context.

Context	Valid	Transitions
Uninitialized	none	htInit (Initialized)
Initialized	Global, Addin, Server, Database, TextList, Datetime, Error	htCurOpen (Open Cursor) htTerm (Uninitialized)
Open Cursor	Cursor, Form, View, Field, Column, Macro, Formula, Mail	htCurClose (Initialized) htItemBind (Bound Item) htFormulaExec (Active Index) htDocOpen, htDocCreate (Open Document)
Bound Item	htDocPut	htCurReset (Open Cursor), htFormulaExec (Active Index)
Active Index	Index, Cell*	htCurReset (Open Cursor) htCellBind* (Active Index/Bound Cell) htItemBind (Active Index/Bound Item)
Active Index / Bound Cell	htDocFetch	htCurReset (Open Cursor) htIndexSearch, htFormulaExec (Active Index) htItemBind (Active Index/Bound Item)
Active Index / Bound Item	htDocPut	htIndexSearch, htFormulaExec (Active Index) htDocFetch (Fetched Document)
Open Document	Document, Item, File	htDocClose (Open Cursor), htItemFetch/htItemGetPtr to type HTTYPE_COMPOSITE (Open Composite) htCompImport, htCompCreate (Open Composite)
Fetched Document	Document, Item, File, htDocUpdate	htDocClose (Active Index/Bound Item) htIndexNavigate, htIndexSetPos, htIndexSetTreePos (Active Index/Bound Item) htIndexSearch, htFormulaExec (Active Index) htItemFetch/htItemGetPtr to type HTTYPE_COMPOSITE (Open Composite) htCompImport, htCompCreate (Open Composite)
Open Composite	Composite, Comprecord	none

** Cell functions are only valid in view-based indices.*

Context Transition Table

In HiTest, higher contexts contain lower contexts. The following diagram shows context containment.



Context Containment Diagram

Program context affects the method of data access. Often data is accessible with either a Fetch operation or a GetPtr operation. The Fetch operation transfers data to a buffer supplied by the calling program. When the data source becomes invalid (e.g., by closing the document), this data is still valid since it is a copy into memory managed by the calling program. The GetPtr operation simply returns a pointer to memory managed by HiTest. The GetPtr operation is more efficient than the Fetch operation when retrieving the same data value repeatedly. When the data source becomes invalid in this case, the data pointer becomes invalid. The calling program can no longer access the data, and should not free the data pointer itself since HiTest manages the memory.

3.5 Error Handling

The HiTest API supports three styles of error handling, so developers can implement error handling in the manner with which they are most familiar and comfortable. HiTest handles errors on a process level (i.e., errors in multiple cursors in a single process are stored in a single location). When an error occurs, there are three pieces of information available: the error code, the error severity, and the error string. The next HiTest API call automatically clears error information (except htError functions, which do not affect the current error information). For more detail on error functions, see the htError function descriptions.

The first style of error handling is the most basic. Most HiTest functions return an HTSTATUS value representing the error code. When this value indicates an error condition, use the htErrorFetch function to get more error information. Error return values are easily checked by comparing them against zero since successful operation (HTSUCCESS) is always equal to zero, and nonzero indicates an error condition.

The second style of error handling involves assigning writeback buffers with htErrorSetBuffer. The calling program assigns buffers to receive one or more of the error code, severity, and string when an error occurs. Just before returning, each HiTest function checks for an error condition and automatically writes any error information to the error buffers. Return values may still be used to check status.

The third style of error handling assigns a callback function with htErrorSetProc. The calling program defines a callback function to be called by HiTest when an error occurs. Declare this function to match the HTERRORPROC type. When invoked, the callback function receives as input parameters the error code, error severity, and error string pointer. In addition, HiTest passes a parameter, supplied by the calling program to htErrorSetProc, to the callback function. This supplies access to information and context from the calling program within the callback function. HiTest functions call the callback function as the last action before returning an error code. If a program uses both this method and writeback buffers (they are usually used exclusively), HiTest writes to the writeback buffers before calling the callback function. Return values may still be used to check status.

A summary of HiTest errors and severities follows:

Class	Error constant	Description
General	HTFAIL_NOTES_ERROR	Standard Notes API error
	HTFAIL_PROGRAM	Software error -- contact Edge Research technical support
	HTFAIL_ILLEGAL_ENUM	An enumeration parameter value is invalid
	HTFAIL_NULL_PARAMETER	A parameter is NULL when a value is required
	HTFAIL_OVERFLOW	A data value cannot fit in the supplied buffer
	HTFAIL_DUPLICATE	Name is already in use
	HTFAIL_BAD_FORMAT	A parameter value is formatted incorrectly
	HTFAIL_END_OF_DATA	The last data value has been retrieved
	HTFAIL_DATA_UNAVAIL	Requested data is unavailable
Global	HTFAIL_INCORRECT_DLL	Program compiled with an incompatible version of HiTest
	HTFAIL_ALREADY_INIT	htInit called when HiTest is already initialized
	HTFAIL_NOT_INIT	HiTest function other than htInit called when not initialized
	HTFAIL_INVALID_CONVERT	The requested conversion is not legal

Db	HTFAIL_INVALID_DATABASE	No such database exists
Cursor	HTFAIL_INVALID_CURSOR	The cursor is invalid
	HTFAIL_OPEN_DOCUMENTS	htCurClose called without force and documents are open
	HTFAIL_ACTIVE_RESULT	An active result prevents the requested operation
Form	HTFAIL_INVALID_FORM	No such form exists in the database
	HTFAIL_FORM_UNAVAIL	Strict binding is on, but no active form is set
View	HTFAIL_INVALID_VIEW	No such view exists in the database
Field	HTFAIL_INVALID_FIELD	No such field exists in the form
Column	HTFAIL_INVALID_COLUMN	No such column exists in the view
Macro	HTFAIL_INVALID_MACRO	No such macro exists in the database
Formula	HTFAIL_INVALID_FORMULA	The formula is invalid
Index	HTFAIL_INVALID_NAVTYPE	View-style navigation attempted on a flat (non-view) index
Doc	HTFAIL_INVALID_DOCUMENT	The document handle or ID is invalid
Item	HTFAIL_INVALID_ITEM	No such item exists in the document
File	HTFAIL_INVALID_DIRECTORY	No such directory exists
	HTFAIL_INVALID_FILE_ITEM	No such file attachment exists in the document
Comp	HTFAIL_INVALID_COMPOSITE	Invalid composite handle
	HTFAIL_INVALID_IMPEXP	No such import/export format exists
	HTFAIL_INVALID_FONT	No such font exists in the document font table

Severity	Description
HTSEVERITY_NOERROR	No error
HTSEVERITY_WARNING	Warning-level error
HTSEVERITY_NONFATAL	Normal error
HTSEVERITY_USAGE	Error in calling program's usage
HTSEVERITY_FATAL	Fatal error
HTSEVERITY_PROGRAM	Internal software error - contact Edge Research

3.6 Mixing HiTest with the standard Notes API

The HiTest API is sufficient to perform the vast majority of Notes API programming. There are two situations, though, where a program needs some mixture of standard Notes API code and HiTest API code. The first occurs when migrating or expanding a program written against the standard Notes API to HiTest. The second occurs when an API program needs some of the more esoteric Notes API functionality not available with HiTest, such as user registration. In either case, the mixture is straightforward as long as a few guidelines are followed:

8. Replace NotesInit and NotesTerm calls with htInit and htTerm calls. These calls include embedded NotesInit and NotesTerm calls.
9. Obtain the standard Notes API database or document handle for a HiTest cursor or document with the GetInfo functions (i.e., htCurGetInfo or htDocGetInfo). With the Notes API handle programs can use the standard Notes API to manipulate an object opened with HiTest.
10. Use care when manipulating the same object (database, document, etc.) with both APIs. For example, a document opened in HiTest can be manipulated through the standard Notes API by obtaining the NOTEHANDLE with htDocGetInfo. Using the standard Notes API to append an item is legal, but using the standard API to delete the document is not legal.

4. HiTest Functions

4.1 Overview

The HiTest API divides functions into functional groupings by object. The actions to perform are verbs and are generally consistent between objects. Each function name consists of four parts, describing the function with an object, a verb, and an optional modifier. The first part is always the prefix ‘ht’. The second part is the object name. Functions in the global grouping omit this part. The third part is the verb. The fourth part, only sometimes used, is the modifier. Taking htDocGetInfo as an example, the object is ‘Doc’, the verb is ‘Get’, and the modifier is ‘Info’.

The following summary tables describe the objects and verbs with modifiers.

Objects	Verbs used with modifiers
(Global)	Init, Term, SetOption, GetInfo, GetEnvString, SetEnvString, ConvertLength, Convert
Addin	SetStatus, PutMsg, SetInterval, GetInterval, Yield
Server	List, GetInfo, Exec
Database	List, ListCat, GetPath
Cursor	Open, Close, GetInfo, SetOption, Reset
Form	List, GetId, GetAttrib, Copy, Delete, Set, Template
View	List, GetId, GetAttrib, Copy, Delete, Set
Field	Count, List, GetInfo
Column	Count, List
Macro	List, GetId, Copy, Delete, Exec
Formula	Concat, Concatf, Length, Copy, Reset, Exec
Index	GetInfo, Count, Navigate, GetPos, SetPos, GetTreePos, SetTreePos, Refresh, Search
Document	Fetch, Put, Copy, Open, Close, Create, GetInfo, Update, Delete
Item	Bind, Unbind, Count, List, GetInfo, Length, Fetch, GetPtr, Put, Delete
Cell	Bind, Unbind, Length, Fetch
File	List, Fetch, Put, Delete
Mail	Send
Composite	GetInfo, Merge, Create, Copy, CopySubset, ListText, ImportList, ExportList, Import, Export, GetOSFont, PutOSFont
Comprec	Count, List, Insert, Update, Delete, Length, Fetch, GetPtr
TextList	Count, Length, Fetch, GetPtr
Datetime	Create, GetInfo, Compare, Diff, Update
Error	Fetch, SetBuffer, SetProc

Verbs	Modifiers Used
Get	EnvString, Interval, Path, Info, Id, Attrib, Pos, TreePos, Ptr, OSFont
Set	EnvString, Option, Status, Interval, Pos, TreePos, Buffer, Proc
List	Cat, Text
Count	
Open	
Close	
Reset	
Copy	Subset
Create	
Delete	
Convert	Length
Bind	
Unbind	
Exec	
Concat	
Length	
Fetch	
Put	OSFont, Msg
Update	
Merge	
Insert	
Compare	
Diff	
Import	List
Export	List
*Init	
*Term	
*Yield	
*Template	
*Concatf	
*Navigate	
*Refresh	
*Search	
*Send	

** - While most verbs apply to multiple objects, certain objects also support actions specific to that group. Verbs marked with an asterisk are specific to one object.*

HiTest orders function parameters with input parameters first and output parameters last. Most functions return an HTSTATUS type return code, which is HTSUCCESS (zero) for success, and one of the nonzero HTFAIL constant values for failure. These functions put all output values into parameters passed by reference to the function. A few functions return a value when there is no significant failure information beyond a NULL or zero value. These functions' return values are commonly used as input to other functions.

Whenever reasonable, the same verb within different objects uses the same parameters (e.g., most GetInfo function prototypes look similar). Parameter use is also generally consistent across functions. For example, for most functions an input buffer length of zero directs HiTest to determine the length, and an output buffer length of zero indicates the buffer is sufficiently large. Some parameters are optional and allow a NULL or zero value. The function parameter descriptions describe the effects of a NULL or zero value.

4.2 Function Descriptions

The function descriptions are divided into HiTest API functional groupings by object. The objects are listed alphabetically, and the functions alphabetically within their object. The global functions are provided first, since they do not comprise an object, but rather miscellaneous top-level functions. A description of the object itself precedes each group of functions. The object description includes structures, enumerations, and flags relevant to an object's functions. Various objects and functions use fonts, which are described in their own section.

The function descriptions use the following format:

htObjectVerbModifier

Summary One-line summary of the function.

Syntax

```
RETURN_TYPE      htObjectVerbModifier      (parm1_name,
parm2_name);

HTTYPE_PARM1   parm1_name;          /* Input/Output spec
*/

HTTYPE_PARM2   parm2_name;          /* Input/Output spec
*/
```

Description Detailed description of the function.

Parameters

PARM1_NAME

Description of parameter 1.

PARM2_NAME

Description of parameter 2.

Returns RETURN_TYPE with description. Functions returning HTSTATUS values list common failure conditions. This section does not list generic HTSTATUS errors (e.g., HTFAIL_NULL_PARAMETER, HTFAIL_NOT_INIT) since most functions can return these errors.

Example

```
htObjectVerbModifier (parm1, parm2);
```

See Also List of related functions

(Global)

These functions are global within a process or task. These functions have no context beneath the process level (i.e., no other functions affect their operation), and are always usable (after calling htInit). While there are other functions with no context, these functions are in the global classification because they do not apply to any object.

Every HiTest API program must initialize and terminate the HiTest API. Until calling the HiTest initialization function htInit, all other functions will fail. Additionally, every HiTest program must call the htTerm function after all HiTest function calls are complete and before the program terminates. It is crucial to call the termination function to avoid leaving the system in a dangerous state.

The global group contains the following functions:

htConvert	Converts data between data types
htConvertLength	Returns the length of data converted as indicated
htGetEnvString	Retrieves the value of a Notes environment string variable
htGetInfo	Obtains a piece of process-level information
htInit	Initializes the HiTest API
htSetEnvString	Assigns the value of a Notes environment string variable
htSetOption	Assigns the value of a global option
htTerm	Terminates the HiTest API

htConvert

Summary Converts data between data types.

Syntax

```
HTSTATUS htConvert (src_type, src_len, src_buffer,
                    dest_type, dest_len, dest_buffer,
                    actual_len);
HTTYPE    src_type;          /* Input */
HTINT          src_len;      /* Input, Optional */
void      *src_buffer;   /* Input */
HTTYPE    dest_type;         /* Input */
HTINT          dest_len;          /*  Input,  Optional
*/
void      *dest_buffer;  /* Output */
HTINT          *actual_len;   /* Output, Optional */
```

Description Converts data between HiTest data types. HiTest writes the converted data into a supplied buffer, and optionally returns the new length. HiTest supports conversion between any data types, with two exceptions. First, HiTest cannot convert between any numeric type (INT, NUMBER, NUMBER_LIST) and any datetime type (DATETIME, TIME_LIST). Second, the only valid conversion involving COMPOSITE is to and from TEXT.

Parameters

SRC_TYPE

The data type of the source data.

SRC_LEN

The length of the source data. A value of zero directs HiTest to determine the length.

SRC_BUFFER

A pointer to the source data.

DEST_TYPE

The data type to convert to in the destination buffer.

DEST_LEN

The length of the destination buffer. A value of zero indicates that the buffer is large enough to hold the result data. A length which is insufficient to contain the result is only valid when the destination type is HTTYPE_TEXT and the global option TEXT_TRUNCATE is active. In this case, the resulting text is truncated to fit in the buffer.

DEST_BUFFER

The destination buffer.

ACTUAL_LEN

If given, this parameter receives the length of the result buffer data.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CONVERT (source type does not convert to destination type);

HTFAIL_OVERFLOW (destination buffer is too small);

HTFAIL_BAD_FORMAT (source data conversion to destination type failed).

Example

```
HTSTATUS status;
HTINT newlen;
HTDATETIME date;
status = htConvert (HTTYPE_TEXT, 0, "Jan 29, 1966",
                    HTTYPE_DATETIME, sizeof (HTDATETIME),
                    &date, &newlen);
```

See Also htConvertLength, htSetOption

htConvertLength

Summary Returns the length of data converted as indicated.

Syntax

```
HTINT htConvertLength (src_type, src_len, src_buffer,
                       dest_type);
HTTYPE    src_type;          /* Input */
HTINT          src_len;      /* Input, Optional */
void      *src_buffer;   /* Input */
HTTYPE    dest_type;         /* Input */
```

Description Determines the length of data resulting from the indicated conversion. Used to determine the buffer length needed when converting to a variable length type. See the htConvert description for a list of invalid conversions.

Parameters

SRC_TYPE

The data type of the source data.

SRC_LEN

The length of the source data. A value of zero directs HiTest to determine the length.

SRC_BUFFER

A pointer to the source data.

DEST_TYPE

The destination data type.

Returns HTINT length of result data. Returns zero for an invalid or illegal conversion.

Example

```
char *string = NULL;
HTINT length;
double number = 3.456;
HTSTATUS htstatus;
```

```
length = htConvertLength (HTTYPE_NUMBER, 0, &number,
                          HTTYPE_TEXT);
if (length != 0)
{
     string = malloc (length + 1);
     status = htConvert (HTTYPE_NUMBER, 0, &number,
                         HTTYPE_TEXT, length, string, NULL);
}
```

See Also htConvert

htGetEnvString

Summary Retrieves the value of a Notes environment string variable.

Syntax

```
HTSTATUS htGetEnvString (name, length, value);
char      *name;          /* Input */
HTINT          length;        /* Input, Optional */
char      *value;         /* Output */
```

Description Retrieves the value of a Notes environment string variable. Notes stores environment variables in the NOTES.INI file. Use the htSetEnvString function to modify Notes environment variables. To retrieve import/export formats, which are stored in multiple environment variables, use the functions htCompImportList and htCompExportList.

Parameters

NAME

The name of the environment variable to retrieve. In the NOTES.INI file, this is the string to the left of the equal sign. Some examples of useful standard Notes environment variables are:

“Directory”	Notes data directory
“MailServer”	Server name on which Notes user’s mailbox resides
“MailFile”	Database filename of Notes user’s mailbox
“Domain”	Notes user’s domain

LENGTH

The length of the value buffer to receive the environment variable value. The constant HTLEN_ENV_STRING defines the maximum length of an environment variable string. A length of zero indicates that the buffer is large enough to hold the result.

VALUE

The buffer to receive the environment variable The constant HTLEN_ENV_STRING defines the maximum length of an environment variable string. In the NOTES.INI file, this is the string to the right of the equal sign.

Returns HTSTATUS return code. Failures include:

HTFAIL_DATA_UNAVAIL (no such environment string).

Example

```
char env_string [HTLEN_ENV_STRING + 1];
HTSTATUS status;
status = htGetEnvString ("VARNAME", 0, env_string);
```

See Also htSetEnvString, htCompImportList, htCompExportList

htGetInfo

Summary Obtains a piece of process-level information.

Syntax

```
HTSTATUS htGetInfo (item, buffer);
HTGLOBINFO     item;          /* Input */
void      *buffer;       /* Output */
```

Description Fetches one of various process-level information items into a supplied buffer. Each item has a data type, and the buffer must be large enough to hold the result.

Parameters

ITEM

One value from an enumeration of global items. Each item corresponds to a type (and length, for variable length types). The following table lists legal items with their corresponding data types and, where relevant, lengths:

constant	type
HTGLOBINFO_USERNAME	char [HTLEN_USERNAME + 1]
HTGLOBINFO_SERVERNAME	char [HTLEN_SERVERNAME + 1]
HTGLOBINFO_CURRENTTIME	HTDATETIME
HTGLOBINFO_TIMEZONE	HTINT
HTGLOBINFO_DST	HTBOOL
HTGLOBINFO_HTVERSION	char [HTLEN_VERSION + 1]

USERNAME obtains the Notes user name in the Notes ID file;

SERVERNAME obtains the name of the locally running Notes server, if any;

CURRENTTIME obtains the current datetime;

TIMEZONE obtains the local time zone as an integer relative to GMT;

DST indicates whether daylight savings time is currently in effect;

VERSION returns the version of HiTest.

BUFFER

The buffer to receive the requested information. This buffer should be of sufficient length to handle the result.

Returns HTSTATUS return code. Failures include:

HTFAIL_ILLEGAL_ENUM (invalid item).

Example

```
char username [HTLEN_USERNAME + 1];
HTSTATUS status;
status = htGetInfo (HTGLOBINFO_USERNAME, username);
```

See Also htInit

htInit

Summary Initializes the HiTest API.

Syntax

```
HTSTATUS htInit (void);
```

Description For each process or task which uses HiTest functions, htInit must be the first HiTest function called. Any other function called before htInit will fail. Note that it is crucial that every HiTest API program invoke both htInit to start and htTerm when complete.

Parameters none.

Returns HTSTATUS return code. Failures include:

HTFAIL_ALREADY_INIT (htInit already called);

HTFAIL_INCORRECT_DLL (program compiled with an incompatible HiTest version).

Example

```
HTSTATUS status;
status = htInit ();
```

See Also htTerm

htSetEnvString

Summary Assigns the value of a Notes environment string variable.

Syntax

```
HTSTATUS htSetEnvString (name, value, create, overwrite);

char     *name;          /* Input */

char     *value;         /* Input */

HTBOOL   create;         /* Input */

HTBOOL   overwrite;          /* Input */
```

Description Assigns a value to a Notes environment string variable. The calling program controls whether to create a new environment variable and whether to overwrite an existing one. Notes stores environment variables in the NOTES.INI file. Use the htGetEnvString function to retrieve Notes environment variables. Use care when modifying Notes environment variables, since they affect the Notes client and server programs.

Parameters

NAME

The name of the environment variable to assign. In the NOTES.INI file, this is the string to the left of the equal sign.

VALUE

The value to assign to the environment variable. The constant HTLEN_ENV_STRING defines the maximum length of an environment variable string. In the NOTES.INI file, this is the string to the right of the equal sign.

CREATE

Whether to create the environment variable if it doesn't exist (TRUE enables creation).

OVERWRITE

Whether to overwrite the environment variable if it does exist (TRUE enables overwrite).

Returns HTSTATUS return code. Failures include:

HTFAIL_DUPLICATE (name exists and overwrite is FALSE);

HTFAIL_DATA_UNAVAIL (name does not exist and create is FALSE);

HTFAIL_OVERFLOW (value is longer than the maximum environment string length).

Example

```
HTSTATUS status;
status = htSetEnvString ("VARNAME", "Variable Value",
TRUE,
                         FALSE);
```

See Also htGetEnvString

htSetOption

Summary Assigns the value of a global option.

Syntax

```
HTSTATUS htSetOption (option, number, string);
HTGLOBOPT option;          /* Input */
HTINT          number;     /* Input, Optional */
char           *string;    /* Input, Optional */
```

Description Assigns a value to a global option or new cursor default. Depending on the option, the new value is supplied in either the number or string parameter. When setting the value of cursor defaults, the new value has no effect on existing cursors.

Parameters OPTION

One value from an enumeration of global options. Each option corresponds to either a string or integer value. The option value indicates whether to use the string or numeric parameter (the function ignores the other parameter). The following table lists legal options with their corresponding data types, parameters, and defaults:

constant	type	parameter	default
HTGLOBOPT_BULK_STORE	HTBOOL	number	FALSE
HTGLOBOPT_STRICT_BIND	HTBOOL	number	TRUE
HTGLOBOPT_VIEW_POSITION	HTBOOL	number	FALSE
HTGLOBOPT_FETCH_SUMMARY	HTBOOL	number	FALSE
HTGLOBOPT_SUMMARY_LIMIT	HTINT	number	8192
HTGLOBOPT_LOCAL_SERVERNAME	char *	string	NULL
HTGLOBOPT_TEXT_TRUNCATE	HTBOOL	number	TRUE

The basic functions of the options are described below:

When bulk store is active, HiTest does not commit changes in document data to disk during a document close, but rather when closing the cursor itself. When bulk store is inactive, closing a document commits all changes to disk.

When strict binding is active, HiTest filters all document items through a form (i.e., HiTest uses the form metadata for type-checking). Therefore, items in a document which are either not in the document's form or are of a different data type than the corresponding field in the document's form will not be accessible. When strict binding is inactive, the items in a document are not tied to the form's metadata. This option also affects certain indices. When executing a formula to produce a flat index and strict binding is active, the index only includes documents of the active form. Strict binding has no effect on the documents included in a view-based index, although it does affect items within documents accessed from a view-based index.

When view position is active, view-based indices always keep depth-first ordinal position information. This enables functions which use index positioning (e.g., htIndexGetPos and htIndexSetPos) to find any element in a view-based index by ordinal location. This functionality may significantly reduce speed. When view position is inactive, HiTest performs view-based positioning off the top-level entries within the index and is much more efficient. This option has no effect on flat indices.

When fetch summary is active, htDocFetch operations open documents with summary data only. Use htDocFetch or the HTDOCHANDLE that it returns to access summary items only. When non-summary items (composite and some other large items) are not needed, this option increases fetch speed. When fetch summary is inactive, documents fetched always have all data available.

The value of summary limit determines the maximum length of an item which will have its summary flag set. If the summary items for a single document exceed the value HTLEN_SUMMARY_DATA, then Notes may not display the document properly in views, and cell values may not be accessible. If the length of any single item exceeds HTLEN_SUMMARY_DATA, then that item is not usable by Notes in any view. The default value of this option is 8K. The constant HTLEN_SUMMARY_DATA (15K) defines the maximum value of this option).

When local servername is set, the string value assigned for the option is usable in place of NULL to indicate the local server. Also, the list of available servers from htServerList includes this string.

When text truncate is active, retrieval of data as text supports truncation of results. Calls to htConvert, htDocFetch, htItemFetch, and htCellFetch with the destination type set to HTTYPE_TEXT will truncate results if necessary, and will not generate an error. Conversions to types other than text do not allow truncation. When text truncate is inactive, no conversions allow truncation.

NUMBER

The numeric or boolean value for an option. When setting the value of a boolean option, use the constants TRUE and FALSE.

STRING

The string value for an option.

Returns HTSTATUS return code. Failures include:

HTFAIL_ILLEGAL_ENUM (invalid option);

HTFAIL_OVERFLOW (value out of bounds).

Example

```
HTSTATUS status;
status = htSetOption (HTGLOBOPT_LOCAL_SERVERNAME, 0,
                      "Local");
```

See Also htConvert, htServerList, htCurOpen, htCurSetOption, htIndexGetPos, htIndexSetPos, htDocFetch, htItemFetch, htCellFetch

htTerm

Summary Terminates the HiTest API.

Syntax

```
HTSTATUS htTerm (void);
```

Description Shuts down the HiTest API for the current process or task. Before terminating, this function closes any open cursors by calling htCurClose. After calling htTerm, no other HiTest functions may be called except htInit. It is crucial that every HiTest program call htTerm.

Parameters none.

Returns HTSTATUS return code.

Example

```
HTSTATUS status;
status = htTerm ();
```

See Also htInit, htCurClose

Addin

The addin functions are for scheduling, server console control, and message logging. These functions allow a program to exercise some control over a local Notes server console. Additionally, HiTest contains a scheduler which programs can use to schedule actions at periodic intervals. Programs using addin functionality often run on a Notes server, but this is not a requirement. Addin programs are written and run like any other HiTest API program, but use the htAddin functions.

The addin group contains the following functions:

htAddinGetInterval	Polls for scheduling interval events
htAddinPutMsg	Logs an event in the Notes log and to the Notes server console
htAddinSetInterval	Sets a scheduling interval
htAddinSetStatus	Sets a Notes server console task status line
htAddinYield	Yields processor control for a specified period

htAddinGetInterval

Summary Polls for scheduling interval events.

Syntax

```
HTSTATUS    htAddinGetInterval    (wait,    interval,
iteration);

HTBOOL    wait;                /* Input */

HTINT          *interval;          /* Output */

HTINT          *iteration;         /* Output, Optional
*/
```

Description Polls the HiTest scheduler to determine if any assigned intervals have occurred. Programs may set intervals with the htAddinSetInterval function. Each time an interval occurs, a call to htAddinGetInterval succeeds and returns information about the interval. Programs can either poll this function periodically or temporarily surrender program control. When a program surrenders control, HiTest waits for the next interval and then returns. HiTest can manage multiple simultaneous intervals.

When giving control to this function under Windows, messages are passed through to the calling application by HiTest. A WM_QUIT message in this state causes this function to return to the calling application for a normal shutdown. A nested call to this function or htAddinYield while processing a message will fail.

Parameters

WAIT

Whether to wait for the next interval. A value of TRUE directs htAddinGetInterval to yield processing until the next interval occurs. A value of FALSE directs htAddinGetInterval to return immediately, regardless of whether an interval has occurred.

INTERVAL

The buffer to receive the interval which occurred. The interval is the same value set with htAddinSetInterval. If no interval has occurred, HiTest sets this value to zero.

ITERATION

The buffer to receive the execution count for this interval. The first time an interval occurs, HiTest sets this value to one, and then one greater each succeeding interval. This is useful for performing an action after an interval occurs a certain number of times. If no interval has occurred, HiTest sets this value to zero.

Returns HTSTATUS return code. Failures include:

HTFAIL_DATA_UNAVAIL (no intervals set or no interval has occurred);

HTFAIL_END_OF_DATA (Windows nested callback or WM_QUIT received).

Example

```
HTINT interval, iteration;
HTSTATUS status;
status = htAddinSetInterval (60);
while (!htAddinGetInterval (TRUE, &interval, &iteration))
        printf ("\nAnother %ld seconds have passed", interval);
```

See Also htAddinSetInterval, htAddinYield

htAddinPutMsg

OS/2 1.3 only

Summary Logs an event in the Notes log and to the Notes server console.

Syntax

```
HTSTATUS htAddinPutMsg (error, string);
HTBOOL    error;          /* Input */
char      *string;        /* Input */
```

Description Logs a message or error to the Notes log and server console. The message is of the format <DATE> <TIME> <string>:<error>. This function controls the contents of the string and whether to have HiTest append the error string. If there is no locally running Notes server, the message appears on the standard output instead. When using this function, HiTest programs must be built with a STRINGTABLE resource using the constant HTADDIN_RESOURCE_MSG with a resource value of "%s".

Parameters

ERROR

Whether to append the current HiTest error message to the string. A value of TRUE and a valid current error message will append a colon followed by the error message.

STRING

The string to use in the message.

Returns HTSTATUS return code. Failures include:

HTFAIL_BAD_FORMAT (no message data - string is NULL and error is FALSE)

Example

```
HTSTATUS status;
status = htAddinPutMsg (TRUE, "Received an error message");
```

See Also htAddinSetStatus, htErrorFetch

htAddinSetInterval

Summary Sets a scheduling interval.

Syntax

```
HTSTATUS htAddinSetInterval (interval);
HTINT           interval;       /* Input */
```

Description Sets or removes a scheduling interval for the HiTest scheduler. After setting an interval, htAddinGetInterval will successfully return the interval event every “interval” seconds. HiTest can manage multiple simultaneous intervals.

Parameters INTERVAL

The interval to set, in seconds. A negative interval removes the interval (e.g., -5 will remove any 5 interval). An interval of zero clears all intervals.

Returns HTSTATUS return code. Failures include:

HTFAIL_DUPLICATE (the interval exists);

HTFAIL_DATA_UNAVAIL (cannot remove an interval which isn’t set).

Example

```
HTSTATUS status;
status = htAddinSetInterval (60);
```

See Also htAddinGetInterval, htAddinYield

htAddinSetStatus

OS/2 1.3 only

Summary Sets a Notes server console task status line.

Syntax

```
HTSTATUS htAddinSetStatus (show, status);
HTBOOL    show;                /* Input */
char      *status;       /* Input, Optional */
```

Description Controls the contents of any Notes server console status line for this task. When a HiTest program is running on a Notes server, it may create a task status line on the Notes server. This line appears in response to the SHOW TASKS console command (executable with the htServerExec function). The line consists of a program name and status. For a program name to display, build the HiTest program with a STRINGTABLE resource using the constant HTADDIN_RESOURCE_NAME. Notes will display the value of this resource as the program name. This function controls whether to display the status line, and the contents of the status part of the line. HiTest removes any status line on termination.

Parameters

SHOW

Whether to show a status line on the Notes server console. All programs start with no status line. A value of TRUE displays the status line. A value of FALSE removes any displayed status line.

STATUS

The status value to display. HiTest ignores this parameter if the show parameter is FALSE. To display a status line but no status value, use NULL or the empty string.

Returns HTSTATUS return code. Failures include:

HTFAIL_DATA_UNAVAIL (HiTest cannot set the status line).

Example

```
HTSTATUS status;
status = htAddinSetStatus (TRUE, "Addin task status
line");
```

See Also htAddinPutMsg, htServerExec, htTerm

htAddinYield

Summary Yields processor control for a specified period.

Syntax

```
HTSTATUS htAddinYield (delay_msec);
HTINT          delay_msec;          /* Input */
```

Description Suspends processing of the current task or process for a specified time. This functionality is also available within the htAddinGetInterval function, in the form of waiting for a scheduler interval. When running under Windows and not using the wait option of htAddinGetInterval, programs should call this function periodically with a delay of zero to allow other tasks to run.

When giving control to this function under Windows, messages are passed through to the calling application by HiTest. A WM_QUIT message in this state causes this function to return to the calling application for a normal shutdown. A nested call to this function or htAddinGetInterval while processing a message will fail.

Parameters DELAY_MSEC

The number of milliseconds to wait before resuming processing. A value of zero will surrender only the current timeslice.

Returns HTSTATUS return code. Failures include:

HTFAIL_BAD_FORMAT (negative delay is invalid);

HTFAIL_END_OF_DATA (Windows nested callback or WM_QUIT received).

Example

```
HTSTATUS status;
status = htAddinYield (1000);
```

See Also htAddinSetInterval, htAddinGetInterval

Cell

A Notes view contains both metadata and data. The metadata part is a set of columns. The data part is a set of cells. Each row in a view represents a document, a category, or totals. The overall view data creates an NxM table of data cells, where N is the number of columns and M is the number of documents. Notes computes these read-only cells from document data. The primary attributes of a cell are a column number, a view row, and a data value.

To render cell data into a view resembling the Notes UI format, programs must follow certain guidelines. The Notes UI normally truncates the data for a given cell at the right boundary of the cell's column. In two situations, though, a cell's data overruns the right boundary, and no more cell data exists for the row. The first case is for response documents in response-only columns (indicated by the flags field in the column attributes structure). The second case is for category rows (indicated by the document ID returned from htIndexNavigate. Additionally, when a view row is beneath the top level (i.e., a response document or a cascading category), the indent returned from htIndexNavigate indicates the number of indentation levels. The Notes UI represents each indentation level with three spaces preceding the row's data. Finally, if a column's attributes have the ICON flag set, then the cell data indicates the Notes icon to use in place of the data value. There are five icons, represented by the values "1" through "5", which are described in the Lotus Notes application documentation.

The cell group contains the following functions:

htCellBind	Binds a cell column to a program variable
htCellFetch	Converts and retrieves the data for a cell into a supplied buffer
htCellLength	Obtains the length of a cell as converted to a specified data type
htCellUnbind	Removes the binding of a cell column

htCellBind

Summary Binds a cell column to a program variable.

Syntax

```
HTSTATUS htCellBind (cursor, column, type, length, buffer,
                     datalen);
HTCURSOR  cursor;        /* Input */
HTINT          column;        /* Input */
HTTYPE    type;          /* Input, Optional */
HTINT          length;        /* Input, Optional */
void      *buffer;       /* Input */
HTINT          *datalen;          /* Input, Optional */
```

Description Creates a 'binding' between a variable in the calling program and a cell / view column in the active view-based index. Use htDocFetch to fetch data for bound cells from the current entry in a view-based index. Use cell binding when fetching the same set of cells from multiple view rows. Create cell bindings after using htFormulaExec to produce a view-based index. Remove cell bindings with htFormulaExec, htCurReset, or htCellUnbind with the same column number. Fetching a document causes all bound cells to be converted and transferred from the current view row to the bound buffers. Unlike bound items, cell binding is not relevant for data storing (htDocPut and htDocUpdate).

Parameters

CURSOR

The cursor containing the desired index.

COLUMN

The column number to bind. The first column number in a view is one.

TYPE

The data type for data in the supplied buffer. When fetching, HiTest converts cell data to this type before writing it into the buffer. This enables automatic conversion between a cell's data and the supplied buffer. A value of zero directs HiTest to use the type HTTYPE_TEXT.

LENGTH

The maximum length of the supplied buffer. Use zero when supplying a buffer known to be of sufficient length.

BUFFER

The buffer into which to copy the cell data.

DATALEN

The buffer to receive the fetched cell data length. When fetching a cell, HiTest sets this value to the actual length of the data retrieved. Use NULL to omit this functionality.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_DATA_UNAVAIL (no active view-based index);

HTFAIL_INVALID_COLUMN (column number is out of range).

Example

```
char date_string [HTLEN_DATETIME_TEXT + 1];
HTSTATUS status;
status = htCellBind (cursor, 1, HTTYPE_TEXT, 0,
date_string,
                    NULL);
```

See Also htCellUnbind, htFormulaExec, htDocFetch, htItemBind, htCurReset

htCellFetch

Summary Converts and retrieves the data for a cell into a supplied buffer.

Syntax

```
HTSTATUS htCellFetch (cursor, column, type, length, buffer);

HTCURSOR  cursor;        /* Input */

HTINT          column;        /* Input */

HTTYPE    *type;         /* Input/Output, Optional */

HTINT          *length;       /* Input/Output, Optional */

void      *buffer;       /* Output */
```

Description Transfers the cell's data from the current view-based index entry to a supplied buffer. If requested, HiTest converts the data before writing it to the buffer. Use htCellLength to determine the required buffer length.

Parameters

CURSOR

The cursor containing the view-based index.

COLUMN

The column number from which to retrieve cell data.

TYPE

The data type representing the destination type -- HiTest converts the cell data to this type before writing it into the supplied buffer. A value directs HiTest to use the type HTTYPE_TEXT.

LENGTH

The length of the supplied buffer. A value zero or a NULL pointer indicates that the buffer is large enough to hold the result data. A value of zero directs HiTest to return the retrieved data length in this location. Use htCellLength to determine the length before retrieving the data. A length which is insufficient to contain the result is valid only when the destination type is HTTYPE_TEXT and the global option TEXT_TRUNCATE is active. In this case, the resulting text is truncated to fit in the buffer.

BUFFER

The buffer to receive the converted data value.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_DATA_UNAVAIL (no active view-based index);

HTFAIL_INVALID_COLUMN (column number is out of range);

HTFAIL_INVALID_CONVERT (cell type does not convert to requested type);

HTFAIL_OVERFLOW (retrieved data does not fit in supplied buffer).

Example

```
HTINT length = 0;
HTTYPE type = HTTYPE_DATETIME;
HTDATETIME datetime;
HTSTATUS status;
status = htCellFetch (cursor, 1, &type, &length,
&datetime);
```

See Also htCellLength, htFormulaExec, htIndexNavigate

htCellLength

Summary Obtains the length of a cell as converted to a specified data type.

Syntax

```
HTSTATUS htCellLength (cursor, column, type, length);
HTCURSOR  cursor;        /* Input */
HTINT          column;        /* Input */
HTTYPE    *type;         /* Input/Output, Optional */
HTINT          *length;       /* Output */
```

Description Obtains the length of a cell's data from the current view-based index entry as converted to a specified data type. Use this length to allocate a buffer of the proper length for htCellFetch.

Parameters

CURSOR

The cursor used containing the view-based index.

COLUMN

The column number for which the length is determined.

TYPE

The data type representing the destination type -- the length returned is the length of the cell data as converted to this type. A value directs HiTest to use the type HTTYPE_TEXT.

LENGTH

The buffer to receive the data length. This is the length of the data as converted to the requested type.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_DATA_UNAVAIL (no active view-based index);

HTFAIL_INVALID_COLUMN (column number is out of range);

HTFAIL_INVALID_CONVERT (cell type does not convert to requested type).

Example

```
HTINT length;
```

```
HTSTATUS status;
status = htCellLength (cursor, 1, HTTYPE_TEXT,
&length);
```

See Also htCellFetch

htCellUnbind

Summary Removes the binding of a cell column.

Syntax

```
HTSTATUS htCellUnbind (cursor, column);
HTCURSOR  cursor;         /* Input */
HTINT           column;         /* Input */
```

Description Cancels the effects of any htCellBind performed with the same column number. Producing a new index automatically cancels all bindings.

Parameters

CURSOR

The cursor used in the binding operation.

COLUMN

The column number used in the binding operation.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_INVALID_COLUMN (column number is not bound).

Example

```
HTSTATUS status;
status = htCellUnbind (cursor, 1);
```

See Also htCellBind, htCurReset

Column

A column defines one data column and optionally one index within a view. Notes produces rows (documents, categories, and totals) when computing a view. The meeting of one column and one row is a cell. The primary attributes of a column are a title and index.

The following flags define column attributes in the HTCOLUMN structure:

HTCOLUMN_SORT	Column is sorted (indexed)
HTCOLUMN_SORT_CAT	Column is a category
HTCOLUMN_SORT_DESC	Sort descending (default is ascending)
HTCOLUMN_HIDDEN	Column is hidden
HTCOLUMN_RESPONSE	Column is a response-only column
HTCOLUMN_HIDE_DETAIL	Hide detail on subtotaled columns
HTCOLUMN_ICON	Display icon instead of text
HTCOLUMN_JUSTIFY_RIGHT	Right justify (default is left)
HTCOLUMN_JUSTIFY_CENTER	Center justify (default is left)

The column attributes may include no more than one of the following flags:

HTCOLUMN_TOTAL	Total all values
HTCOLUMN_AVG_PER_CHILD	Average per child
HTCOLUMN_PCT_OF_VIEW	Percent of total view
HTCOLUMN_PCT_OF_PARENT	Percent of parent category
HTCOLUMN_AVG_PER_DOC	Average per document

Use the HTCOLUMN_MASK_TOTAL constant to exclude non-totals attributes from column flags (e.g., flags & HTCOLUMN_MASK_TOTAL).

htColumnList returns the following column attribute structure:

```
typedef struct
{
   char name [HTLEN_COLUMNNAME + 1];     /* Column name */
   HTINT width;                 /* Display width in 1/8 avg. char
                                   units */
   HTFLAGS flags;                 /* Column flags (HTCOLUMN_xxx)
*/
   HTFONT font;                 /* Column font information */
} HTCOLUMN;                          /* Column attribute structure
*/
```

The column group contains the following functions:

htColumnCount	Obtains the number of columns in a view
htColumnList	Iterates through columns in a view

htColumnCount

Summary Obtains the number of columns in a view.

Syntax

```
HTSTATUS htColumnCount (cursor, viewid, colcount);
HTCURSOR  cursor;        /* Input */
HTVIEWID  viewid;        /* Input */
HTINT          *colcount;          /* Output */
```

Description Obtains the number of columns in the indicated view.

Parameters

CURSOR

The cursor containing the view.

VIEWID

The view from which to obtain the column count.

COLCOUNT

The buffer to receive the number of columns in the view.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_INVALID_VIEW (view does not exist).

Example

```
HTINT colcount;
HTSTATUS status;
status = htColumnCount (cursor, viewid, colcount);
```

See Also htViewGetId, htColumnList

htColumnList

Summary Iterates through columns in a view.

Syntax

```
HTSTATUS htColumnList (cursor, viewid, first, column);
HTCURSOR  cursor;        /* Input */
HTVIEWID  viewid;        /* Input */
HTBOOL    first;         /* Input */
HTCOLUMN  *column;       /* Output */
```

Description Returns the first or next column information from the list of columns in the view.

Parameters

CURSOR

The cursor containing the view.

VIEWID

The view from which to list columns.

FIRST

Whether to get the first or next column. TRUE resets the column list, FALSE obtains the next column in the list. This value is always TRUE on the first call for a given view.

COLUMN

The structure to receive information on the column. See the Column object section preceding the htColumn functions for a description of this structure and its contents.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_INVALID_VIEW (view does not exist);

HTFAIL_END_OF_DATA (no more columns).

Example

```
HTCOLUMN column;
HTSTATUS status;
status = htColumnList (cursor, FALSE, &column);
```

See Also htViewGetId, htColumnCount

Composite

Composite objects are a special free-form data type within Lotus Notes. Each composite consists of one or more subcomponents called composite records. Internally, Notes stores a composite object as one or more items within a document. When using multiple items, the name is the same for all the items, and HiTest handles and presents them as a single item. The primary attributes of a composite are an item name, a handle, and a value. HiTest uses the constant NULLHANDLE to represent an invalid composite handle. Composite data is synonymous with rich text or compound text.

A composite handle represents an existing or new composite item. Obtain a handle to an existing composite item by retrieving the item's value with htDocFetch, htItemFetch, or htItemGetPtr. The composite handle is the value retrieved by these functions. Certain composite functions (htCompCreate, htCompCopy, htCompCopySubset, and htCompImport) create a new composite item in a supplied document. Normally, when closing a document containing one or more open composite handles, HiTest stores the new composite values in the document (unless discarding the changes -- see htDocClose). Alternatively, a composite item created with the htComp functions with a NULL or empty item name is a temporary composite value discarded when the composite handle becomes invalid. This functionality is useful as a composite scratchpad or when manipulating composite data from a composite item considered read-only. HiTest supports multiple nameless composite items within a document (i.e., the composite handles are different for each).

The composite group contains the following functions:

htCompCopy	Creates a new composite whose contents are a copy of another composite
htCompCopySubset	Creates a new composite whose contents are a partial copy another composite
htCompCreate	Creates a new, empty composite within a document
htCompExport	Exports a composite to a file in a selected format
htCompExportList	Iterates through available composite export formats
htCompGetInfo	Obtains a piece of information from and about an open composite
htCompGetOSFont	Obtains operating system-specific font information from a Notes font
htCompImport	Imports and converts a file into a new composite
htCompImportList	Iterates through available composite import formats
htCompListText	Iterates through text in a composite.
htCompMerge	Merges the contents of one composite into another composite
htCompPutOSFont	Generates a Notes font from operating system-specific font information

htCompCopy

Summary Creates a new composite whose contents are a copy of another composite.

Syntax

```
HTSTATUS htCompCopy (src_comphand, dest_dochand,
itemname,
                     dest_comphand);
HTCOMPHANDLE   src_comphand;  /* Input */
HTDOCHANDLE         dest_dochand;  /* Input */
char           *itemname;          /*  Input,  Optional
*/
HTCOMPHANDLE   *dest_comphand;     /* Output */
```

Description Creates a new composite within a document from the data in an existing composite. The new composite becomes an item in the document with the given item name. Creating a composite with no item name results in a temporary composite that is destroyed when closing the document.

Parameters

SRC_COMPHAND

The composite to copy.

DOCHAND

The document in which to create the new composite.

ITEMNAME

The item name for the new composite. An item name of NULL or the empty string creates a temporary composite for use as a scratchpad. Closing the containing document destroys a temporary composite.

NEWCOMP

The buffer to receive the handle to the new composite.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_COMPOSITE (invalid composite handle);

HTFAIL_INVALID_DOCUMENT (invalid document handle);

HTFAIL_INVALID_FIELD (field is not in the document's form).

Example

```
HTCOMPHANDLE newcomp;
HTSTATUS status;
status = htCompCopy (oldcomp, dochand, "NewBody",
&newcomp);
```

See Also htCompCreate, htCompCopySubset

htCompCopySubset

Summary Creates a new composite whose contents are a partial copy of another composite.

Syntax

```
HTSTATUS htCompCopySubset (src_comphand, start, count,
                           dest_dochand, itemname,
                           dest_comphand);
HTCOMPHANDLE   src_comphand;  /* Input */
HTINT               start;         /* Input */
HTINT               count;         /* Input */
HTDOCHANDLE         dest_dochand;  /* Input */
char           *itemname;          /*  Input,  Optional
*/
HTCOMPHANDLE   *dest_comphand;     /* Output */
```

Description Creates a new composite within a document from part of the data in an existing composite. The new composite becomes an item in the document with the given item name. Creating a composite with no item name results in a temporary composite destroyed when closing the document.

Parameters

SRC_COMPHAND

The composite to copy.

START

The composite record index in the source composite at which to begin the data copy. The first composite record is index one. Use the htComprecCount function to obtain the last index for a particular composite.

COUNT

The number of composite records to copy from the source composite.

DOCHAND

The document in which to create the new composite.

ITEMNAME

The item name for the new composite. An item name of NULL or the empty string creates a temporary composite for use as a scratchpad. Closing the containing document destroys a temporary composite.

NEWCOMP

The buffer to receive the handle to the new composite.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_COMPOSITE (invalid composite handle);

HTFAIL_INVALID_DOCUMENT (invalid document handle);

HTFAIL_INVALID_FIELD (field is not in the document's form);

HTFAIL_END_OF_DATA (index is invalid).

Example

```
HTCOMPHANDLE newcomp;
HTSTATUS status;
status = htCompCopySubset (oldcomp, 1, 20, dochand,
                           "NewBody", &newcomp);
```

See Also htCompCopy, htComprecCount

htCompCreate

Summary Creates a new, empty composite within a document.

Syntax

```
HTSTATUS htCompCreate (dochand, itemname, newcomp);
HTDOCHANDLE         dochand;        /* Input */
char           *itemname;           /*  Input,  Optional
*/
HTCOMPHANDLE   *newcomp;            /* Output */
```

Description Creates a new empty composite within a document. The new composite becomes an item in the document with the given item name. Creating a composite with no item name results in a temporary composite destroyed when closing the document.

Parameters

DOCHAND

The document in which to create the composite.

ITEMNAME

The item name for the new composite. An item name of NULL or the empty string creates a temporary composite for use as a scratchpad. Closing the containing document destroys a temporary composite.

NEWCOMP

The buffer to receive the handle to the new composite.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_DOCUMENT (invalid document handle);

HTFAIL_INVALID_FIELD (field is not in the document's form).

Example

```
HTCOMPHANDLE comphand;
HTSTATUS status;
status = htCompCreate (dochand, "NewBody", &comphand);
```

See Also htCompGetInfo, htCompCopy, htCompImport

htCompExport

Summary Exports a composite to a file in a selected format.

Syntax

```
HTSTATUS htCompExport (comphand, filename, format);
HTCOMPHANDLE   comphand;           /* Input */
char           *filename;          /* Input */
char           *format;       /* Input */
```

Description Creates a file from a composite object. HiTest supports export to any export format normally supported by Notes.

Parameters

COMPHAND

The composite to export.

FILENAME

The file in which the exported data goes. Use the fully specified path if the file is not in the current working directory.

FORMAT

The composite export format to use for conversion. Use htCompExportList to obtain a list of available export formats.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_COMPOSITE (invalid composite handle);

HTFAIL_INVALID_IMPEXP (invalid export format).

Example

```
HTSTATUS status;
status = htCompExport (comphand, "filename.rtf",
                       "MicrosoftWord RTF");
```

See Also htCompExportList, htCompImport

htCompExportList

Summary Iterates through available composite export formats.

Syntax

```
HTSTATUS htCompExportList (first, format, extensions);
HTBOOL    first;          /* Input */
char      *format;        /* Output */
char      *extensions;    /* Output, Optional */
```

Description Returns the first or next composite export format information from the list of composite export formats.

Parameters

FIRST

Whether to get the first or next composite export format information. TRUE resets the export format list, FALSE simply obtains the next export format in the list. The value is always TRUE on the first call after htInit.

FORMAT

The buffer to receive the composite export format string. Each format string defines one export format for the htCompExport function. The constant HTLEN_IMPEXPINFO defines the maximum format string length.

EXTENSIONS

The buffer to receive the composite export format file extensions. Commas separate the extensions, which include the period (e.g., ".txt,.asc"). The constant HTLEN_IMPEXPINFO defines the maximum extension string length.

Returns HTSTATUS return code. Failures include:

HTFAIL_END_OF_DATA (no more export formats).

Example

```
char format [HTLEN_IMPEXPINFO + 1];
char extensions [HTLEN_IMPEXPINFO + 1];
HTSTATUS status;
status = htCompExportList (FALSE, format, extensions);
```

See Also htCompExport, htCompImportList

htCompGetInfo

Summary Obtains a piece of information from and about an open composite.

Syntax

```
HTSTATUS htCompGetInfo (comphand, item, buffer);
HTCOMPHANDLE   comphand;           /* Input */
HTGLOBINFO          item;          /* Input */
void           *buffer;       /* Output */
```

Description Fetches one of various composite-level information items into a supplied buffer. Each item has a data type and the buffer must be sufficiently large to hold the result.

Parameters

COMPHAND

The composite on which to obtain information.

ITEM

One value from an enumeration of composite items. Each item corresponds to a type. The following table lists legal items with their corresponding data types and, where relevant, lengths:

constant	type
HTCOMPINFO_ITEMNAME	char [HTLEN_FIELDNAME + 1]
HTCOMPINFO_ISDIRTY	HTBOOL
HTCOMPINFO_HTDOCHANDLE	HTDOCHANDLE
HTCOMPINFO_HTCURSOR	HTCURSOR

ITEMNAME obtains the name of the composite item in the document.

ISDIRTY obtains a boolean which indicates whether data has been altered;

HTDOCHANDLE obtains the HiTest document handle containing this composite;

HTCURSOR obtains the HiTest cursor in which this composite's document was opened.

BUFFER

The buffer to receive the requested information. This buffer should be of sufficient length to handle the result.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_COMPOSITE (invalid composite handle);

HTFAIL_ILLEGAL_ENUM (invalid item).

Example

```
char compitem [HTLEN_FIELDNAME + 1];
HTSTATUS status;
status = htCompGetInfo (comphand, HTCOMPINFO_ITEMNAME,
                        compitem);
```

See Also htItemFetch, htDocOpen, htDocFetch, htCompCreate

htCompGetOSFont

Windows only

Summary Obtains operating system-specific font information from a Notes font.

Syntax

```
HTSTATUS htCompGetOSFont (comphand, htfont, osfont);
HTCOMPHANDLE   comphand;           /* Input */
HTFONT         *font;          /* Input */
OSFONT         *osfont;        /* Output */
```

Description Given a Notes font, obtains operating system-specific font information. HiTest represents OS-specific information with an OS-specific structure.

Parameters

COMPHAND

The composite containing the font.

FONT

The Notes font about which to obtain operating system-specific information. If the font is not a standard Notes font, HiTest obtains the information from an internal font item in the composite's document.

OSFONT

The operating system-specific font information. The OSFONT is an operating system-specific typedef. Currently only the Windows version of HiTest supports this function, which defines OSFONT as LOGFONT.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_COMPOSITE (invalid composite handle);

HTFAIL_INVALID_FONT (cannot find font in document).

Example

```
OSFONT osfont;
HTSTATUS status;
status = htCompGetOSFont (comphand, htfont, &osfont);
```

See Also htCompPutOSFont

htCompImport

Summary Imports and converts a file into a new composite.

Syntax

```
HTSTATUS htCompImport (dochand, itemname, filename,
format,
                       default_font, comphand);
HTDOCHANDLE          dochand;        /* Input */
char            *itemname;           /*  Input,  Optional
*/
char            *filename;           /* Input */
char            *format;        /* Input */
HTFONT          *default_font;       /*  Input,  Optional
*/
HTCOMPHANDLE    *comphand;           /* Output */
```

Description Creates a new composite and imports data from a file into the composite. HiTest supports import to Notes format from any import format normally supported by Notes.

Parameters

DOCHAND

The document in which to create the composite.

ITEMNAME

The item name for the new composite. An item name of NULL or the empty string creates a temporary composite usable as a scratchpad. Closing the containing document destroys a temporary composite.

FILENAME

The file to import. Use the fully specified path if the file is not in the working directory.

FORMAT

The composite import format to use for conversion. Use htCompImportList to obtain a list of available import formats.

DEFAULT_FONT

The default font to use in the new composite. A value of NULL uses the standard default font.

COMPHAND

The buffer to receive the handle to the new composite.

Returns HTSTATUS return code. Failures include:

HTFAIL_NULL_PARAMETER (comphand and itemname are null - no destination);

HTFAIL_INVALID_DOCUMENT (invalid document handle);

HTFAIL_INVALID_FIELD (field is not in the document's form).

HTFAIL_INVALID_IMPEXP (invalid import format).

Example

```
HTCOMPHANDLE comphand;
HTSTATUS status;
status = htCompImport (dochand, "Body", "filename.rtf",
                       "MicrosoftWord RTF", NULL,
                       &comphand);
```

See Also htCompImportList, htCompExport, htCompCreate

htCompImportList

Summary Iterates through available composite import formats.

Syntax

```
HTSTATUS htCompImportList (first, format, extensions);
HTBOOL    first;          /* Input */
char      *format;        /* Output */
char      *extensions;    /* Output, Optional */
```

Description Returns the first or next composite import format information from the list of composite import formats.

Parameters

FIRST

Whether to get the first or next composite import format information. TRUE resets the import format list, FALSE simply obtains the next import format in the list. The value is always TRUE on the first call after htInit.

FORMAT

The buffer to receive the composite import format string. Each format string defines one import format for the htCompImport function. The constant HTLEN_IMPEXPINFO defines the maximum format string length.

EXTENSIONS

The buffer to receive the composite import format file extensions. Commas separate the extensions, which include the period (e.g., ".txt,.asc"). The constant HTLEN_IMPEXPINFO defines the maximum extension string length.

Returns HTSTATUS return code. Failures include:

HTFAIL_END_OF_DATA (no more import formats).

Example

```
char format [HTLEN_IMPEXPINFO + 1];
char extensions [HTLEN_IMPEXPINFO + 1];
HTSTATUS status;
status = htCompImportList (FALSE, format, extensions);
```

See Also htCompImport, htCompExportList

htCompListText

Summary Iterates through text in a composite.

Syntax

```
HTSTATUS htCompListText (comphand, first, word_wrap,
                         tab_spaces, newline, length,
                         buffer, actual_len);
HTCOMPHANDLE   comphand;           /* Input */
HTBOOL         first;         /* Input */
HTINT               word_wrap;          /*       Input,
Optional */
HTINT               tab_spaces;    /*  Input,  Optional
*/
char           *newline;           /*  Input,  Optional
*/
HTINT               length;        /* Input */
char           *buffer;       /* Output */
HTINT               *actual_len;   /*  Output,  Optional
*/
```

Description Returns the first or next text data from a composite. Unlike standard conversion with htConvert, this function can retrieve and apply formatting to all text within a composite by iterating through the text in the composite. Non-text elements in the composite are ignored. This function optionally performs certain text formatting: word wrap, tab replacement with spaces, and a program-defined newline string. Multiple calls to this function for a single composite retrieves all text from the composite.

Parameters

COMPHAND

The composite from which to retrieve text.

FIRST

Whether to get the first or next text. TRUE resets the current text location marker, FALSE simply obtains the next text in the composite. The value is always TRUE on the first call for a given composite handle.

WORD_WRAP

The character position at which to begin a new line. No lines will extend past this position, and this function moves words which run past this position to the next line. A value of zero results in no word wrapping. When using word wrapping, lines of text returned will not be split across calls to this function.

TAB_COUNT

The number of spaces which replace each tab character. A value of zero directs HiTest to strip tabs from text without replacement. A value of HTCOMP_TAB_KEEP directs HiTest to keep tabs in the text buffer rather than replace them.

NEWLINE

The string to use in the retrieved text for each newline. A value of NULL directs HiTest to use the default newline string "\r\n" as the newline delimiter.

LENGTH

The length of the supplied buffer to receive the text.

BUFFER

The buffer to receive the text.

ACTUAL_LEN

The buffer to receive the length of text placed in the buffer. When using word wrapping, this may be up to one line less than the buffer supplied, even though more text is available.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_COMPOSITE (invalid composite handle);

HTFAIL_OVERFLOW (length of zero or less than one line with word wrap is invalid);

HTFAIL_END_OF_DATA (no more text in the composite).

Example

```
char *buffer;
HTINT length;
HTSTATUS status;
buffer = malloc (BUFLEN + 1);
status = htCompListText (comphand, TRUE, 80, 8, NULL,
```

```
                                    BUFLEN, buffer, &length);
```

See Also htConvert, htItemFetch, htItemGetPtr

htCompMerge

Summary Merges the contents of one composite into another composite.

Syntax

```
HTSTATUS htCompMerge (maincomp, addcomp, index);
HTCOMPHANDLE   maincomp;           /* Input */
HTCOMPHANDLE   addcomp;       /* Input */
HTINT               index;         /*  Input,  Optional
*/
```

Description Inserts the contents of one composite into another composite at a specified index. The insertion does not affect the composite being added into the other composite.

Parameters

MAINCOMP

The 'main' composite, into which to merge the added composite (i.e., the composite which is changing).

ADDCOMP

The composite to merge into the main composite.

INDEX

The composite record index in the main composite at which to perform the merge. The first composite record is index one. Use the htComprecCount function to obtain the last index for a particular composite. A value of zero directs HiTest to use the location past the last composite record, resulting in concatenation.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_COMPOSITE (invalid composite handle);

HTFAIL_DUPLICATE (cannot merge a composite into itself);

HTFAIL_END_OF_DATA (index is invalid).

Example

```
HTSTATUS status;
status = htCompMerge (maincomp, addcomp, 0);
```

See Also htComprecCount

htCompPutOSFont

Windows only

Summary Generates a Notes font from operating system-specific font information.

Syntax

```
HTSTATUS htCompPutOSFont (comphand, osfont, htfont);
HTCOMPHANDLE   comphand;           /* Input */
OSFONT         *osfont;        /* Input */
HTFONT         *font;          /* Output */
```

Description Given an operating system-specific font, generates a Notes font. A font not currently used in the composite's document is stored in an internal font item in the document. HiTest represents OS-specific information with an OS-specific structure.

Parameters

COMPHAND

The composite which will contain the font.

OSFONT

The operating system-specific font information. The OSFONT is an operating system-specific typedef. Currently, only the Windows of HiTest supports this function, which defines OSFONT as LOGFONT.

FONT

The buffer to receive the Notes font.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_COMPOSITE (invalid composite handle).

Example

```
HTFONT htfont;
HTSTATUS status;
status = htCompPutOSFont (comphand, osfont, &htfont);
```

See Also htCompGetOSFont

Comprec

Comprec is an abbreviation for composite record. One or more ordered composite records make up a composite. The primary attributes of a composite record are a composite record type, an index, and a value. Currently, HiTest does not support all composite record types available within Lotus Notes, although the common types are supported. Each composite record type is manipulated with a composite record structure prefixed with HTC_.

The following constants define PABDEF justification flags in the HTC_PABDEF structure:

HTPABJUSTIFY_LEFT	Left justification
HTPABJUSTIFY_RIGHT	Right justification
HTPABJUSTIFY_FULL	Full (left and right) justification
HTPABJUSTIFY_CENTER	Center justification
HTPABJUSTIFY_NONE	No word wrap

The following flags define PABDEF flags in the HTC_PABDEF structure:

HTPAB_PAGE_BEFORE	Paragraph starts a new page
HTPAB_KEEP_WITH_NEXT	Keep with next paragraph
HTPAB_KEEP_TOGETHER	Don't split lines in paragraph
HTPAB_PROPAGATE	Propagate PAGE_BEFORE and KEEP_WITH_NEXT
HTPAB_HIDE_READ	Hide paragraph in view mode
HTPAB_HIDE_EDIT	Hide paragraph in edit mode
HTPAB_HIDE_PRINT	Hide paragraph when printing
HTPAB_DISPLAY_RIGHT	Honor right margin when displaying
HTPAB_HIDE_COPY	Hide paragraph when copying/forwarding

Use the constant TWIPS_PER_INCH to manipulate HTC_PABDEF margin fields in inches. A TWIP is a term used in Notes as a fraction (1/1440) of an inch.

The following flags define DDE and OLE clipboard flags in the HTC_DDE and HTC_OLE structures:

HTCLIP_TEXT	Text
HTCLIP_METAFILE	Metafile or MetafilePict
HTCLIP_BITMAP	Bitmap
HTCLIP_RTF	Rich text format

The following flags define DDE flags in the HTC_DDE structure:

HTDDE_HOTLINK	Hot DDE link
HTDDE_WARMLINK	Warm DDE link
HTDDE_ISEMBED	Embedded document is used

The following flags define OLE flags in the HTC_OLE structure:

HTOLE_EMBEDDED	Object is an embedded OLE object
HTOLE_LINK	Object is an OLE link object
HTOLE_HOTLINK	The OLE link object is automatic (hot)
HTOLE_WARMLINK	The OLE link object is manual (warm)

The following table describes the composite record types:

HTCOMP enumeration	Description and structure

PABDEF	A Paragraph Attribute Block Definition. A PABDEF defines the format of a paragraph. While multiple paragraphs may use the same PABDEF, each PABDEF normally occurs only once. After each paragraph begins, a PABREF references the proper PABDEF by its identifier (PABID) field. A PABDEF must come before any references to that PABDEF. The structure for an HTCOMP_PABDEF record is an HTC_PABDEF, and contains no variable length component:
	
	`typedef struct`
	`{`
	`  WORD pabid;  /* ID for this PABDEF */`
	`  WORD justify;  /* Justification method`
	`  (HTPABJUSTIFY_xxx) */`
	`  WORD linespace;  /* 2 * (line spacing - 1) */`
	`  WORD paraspace_before;  /* Linespace units before paragraph */`
	`  WORD paraspace_after;  /* Linespace units after paragraph */`
	`  WORD left_margin;  /* Left margin, in twips (1/1440 inch)`
	`  Notes default is TWIPS_PER_INCH */`
	`  WORD right_margin;  /* Right margin, in twips`
	`  Zero means 1" from right edge */`
	`  WORD first_left_margin;  /* First line left margin, in twips */`
	`  WORD tab_count;  /* Number of tab stops in tabs table */`
	`  short int tabs [20];  /* Tab stops, in twips`
	`  Negative value means decimal tab */`
	`  WORD flags;  /* PABDEF flags (HTPAB_xxx) */`
	`} HTC_PABDEF;  /* Composite record PAB definition */`

PABREF	A Paragraph Attribute Block Reference. This record occurs at the beginning of a paragraph and indicates the PABDEF paragraph definition to use for the current paragraph. A paragraph with no PABREF uses the PABDEF from the previous paragraph. A PABREF contains only an identifier indicating the PABDEF to use. The structure for an HTCOMP_PABREF record is an HTC_PABREF, and contains no variable length component:
	
	`typedef struct`
	`{`
	`  WORD pabid;  /* ID of the PABDEF to reference */`
	`} HTC_PABREF;  /* Composite record PAB reference */`

PARA	An indicator to begin a new paragraph. A PARA does not contain any information or value. The textual representation of a PARA is a newline. There is no structure for an HTCOMP_PARA record, since it contains no data.

TEXT

A run of text. Each TEXT value contains a font and a variable length text component. To change fonts, use a new TEXT record. Notes displays consecutive text runs consecutively (e.g., Notes displays the runs “ABC” and “DEF” as “ABCDEF”). Notes can represent a newline within a text record by a NULL character, but usually uses a paragraph record for a newline. Since embedded NULLs within text runs are valid, a NULL terminator cannot be used to determine string length. When storing a text record, a length of zero indicates no text rather than NULL terminated text. When reading text with htComprecFetch, programs should allocate an extra NULL terminator byte, since HiTest adds a NULL after the text run. This NULL is useful for string manipulation of text runs known to contain no NULLs. The text length does not include this NULL, which does not represent a newline. The structure for an HTCOMP_TEXT record is an HTC_TEXT, and contains a variable length component:

```
typedef struct
{
  HTFONT font;       /* Text font */
  HTINT length;      /* Length of data. Zero indicates no text */
  char *buffer;      /* Pointer to data */
} HTC_TEXT;          /* Composite record text block */
```

BLOB

A catch-all composite record type for unknown composite record types. Manipulation of BLOB contents requires knowledge of the internal structure of composite items defined in the standard Notes API. A BLOB contains variable length data, including the internal composite record signature. Any composite type may be retrieved as a BLOB, which is the only composite conversion allowed. The structure for an HTCOMP_BLOB record is an HTC_BLOB, and contains a variable length component:

```
typedef struct
{
  HTINT length;      /* Length of data (including header) */
  char *buffer;      /* Pointer to data (including header) */
} HTC_BLOB;          /* Generic composite record structure */
```

DOCLINK

A Notes doclink. A doclink is a link to another Notes document, or any replica of that document. The document is represented by a cursor, a view ID, and a document handle of the document to which the link points. Additionally, each doclink contains a comment normally constructed by HiTest (of the form "Database 'Db title', View 'View title', Document 'Document title'"). Programs can override this comment by setting the comment and comment_length structure fields when inserting a new doclink record. When accessing a doclink, the read_comment_only field determines whether to access the comment information only or to access the entire doclink. When accessing the entire doclink (read_comment_only = FALSE), HiTest creates both a cursor and document handle which the calling program must close. HiTest ignores the read_comment_only field when creating new doclinks.

```
typedef struct
{
  HTCURSOR cursor;          /* Cursor containing document */
  HTVIEWID viewid;          /* View to which the link points */
  HTDOCHANDLE dochand;      /* Document to which the link points */
  WORD comment_length;      /* Length of display comment
                               Use zero on input for null term */
  char *comment;            /* Link comment. If omitted on input
                               (empty or NULL), built by HiTest */
  HTBOOL read_comment_only; /* When reading, whether to only
                               retrieve the comment (TRUE), or to
                               open the cursor and document handle
                               (FALSE) */
} HTC_DOCLINK;              /* Composite record doclink */
```

DDE

A DDE link definition and starting point. A DDE link always begins with a DDE record and must end with a DDE_END record. The DDE data is stored between these records as other composite records. A simple DDE record contains values for the server, topic, and item names; the value HTDDE_HOTLINK for flags, and the value HTCLIP_TEXT for clipboard. If a DDE link contains an embedded document (flag HTDDE_ISEMBED flag, embed_count = 1), then the embedded document name is a variable length component at embed_name. The structure for an HTCOMP_DDE record is an HTC_DDE, and contains a variable length component:

```
typedef struct
{
  char server_name [HTLEN_DDE_SERVER];  /* DDE server name */
  char topic_name [HTLEN_DDE_TOPIC];    /* DDE topic name */
  char item_name [HTLEN_DDE_ITEM];      /* DDE item name */
  HTINT flags;                          /* DDE flags (HTDDE_xxx) */
  WORD embed_count;         /* Number of embedded docs (0 or 1) */
  WORD clipboard;           /* Clipboard format (HTCLIP_xxx) */
  WORD embed_length;        /* Length of embedded document name */
  char *embed_name;         /* Embedded document name */
} HTC_DDE;                  /* Composite record DDE object
                               starting point */
```

DDE_END

An indicator to end a DDE object. It is crucial that every DDE record has a matching DDE_END record. A DDE_END does not contain any information or value. There is no structure for an HTCOMP_DDE_END record, since it contains no data.

OLE	An OLE 1.0 link definition and starting point. An OLE link always begins with an OLE record and must end with an OLE_END record. The OLE data is stored between these records as other composite records. An OLE object is either linked or embedded, and if linked is either a hot or warm link. The OLE file is included in the document as a file attachment. The name_buffer field contains the name of the file attachment. When creating a new OLE record, HiTest automatically attaches the specified file to the document (do not use htFilePut). The file name must be given as a fully specified path. HiTest constructs the filename of the attachment using an internally generated unique number different from the filename submitted by the calling program. The generated filename is stored in the composite record. When retrieving an OLE record, obtain the attached file with the htFileFetch function, using the filename stored in the name buffer.
	
	The name buffer field also stores the optional class and template names, if used. The name buffer is a variable length component consisting of 1-3 strings (file name required, class and template names optional) concatenated together. The lengths of each string are defined in the file_length, class_length, and template_length fields. Strings in the name buffer are not NULL terminated. The structure for an HTCOMP_OLE record is an HTC_OLE, and contains a variable length component:
	
	`typedef struct`
	`{`
	`  HTINT flags;  /* OLE flags (HTOLE_xxx) */`
	`  WORD clipboard;  /* Clipboard format (HTCLIP_xxx) */`
	`  WORD file_length;  /* Length of attached file name -`
	`  First string in name_buffer */`
	`  WORD class_length;  /* Length of class name (optional) -`
	`  Second string in name_buffer */`
	`  WORD template_length;  /* Length of template name (optional) -`
	`  Third string in name_buffer */`
	`  char *name_buffer;  /* Name buffer containing 1-3 names,`
	`  lengths given in xxx_length fields */`
	`} HTC_OLE;  /* Composite record OLE object`
	`  starting point */`

OLE_END	An indicator to end an OLE object. It is crucial that every OLE record has a matching OLE_END record. An OLE_END does not contain any information or value. There is no structure for an HTCOMP_OLE_END record, since it contains no data.

The composite record group contains the following functions:

htComprecCount	Obtains the number of composite records in a composite
htComprecDelete	Deletes one or more consecutive composite records from a composite
htComprecFetch	Retrieves the data from a composite record
htComprecGetPtr	Retrieves the values and a data pointer from a composite record
htComprecInsert	Inserts a single composite record into an existing composite
htComprecLength	Obtains the length of the variable length portion of a composite record
htComprecList	Iterates through composite records in a composite

htComprecUpdate	Modifies the contents of a single composite record

htComprecCount

Summary Obtains the number of composite records in a composite.

Syntax

```
HTSTATUS htComprecCount (comphand, count);
HTCOMPHANDLE   comphand;          /* Input */
HTINT              *count;        /* Output */
```

Description Obtains the number of composite records in the composite. Each composite element (e.g., text run, paragraph, etc.) is a single composite record.

Parameters

COMPHAND

The composite from which to obtain the composite record count.

COUNT

The buffer to receive the number of composite records in the composite.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_COMPOSITE (invalid composite handle).

Example

```
HTINT count;
HTSTATUS status;
status = htComprecCount (comphand, &count);
```

See Also htComprecList

htComprecDelete

Summary Deletes one or more consecutive composite records from a composite.

Syntax

```
HTSTATUS htComprecDelete (comphand, start, count);
HTCOMPHANDLE   comphand;          /* Input */
HTINT               start;        /* Input */
HTINT               count;        /* Input */
```

Description Deletes one or more consecutive composite records from a composite.

Parameters

COMPHAND

The composite containing the composite records to delete.

START

The index of the first composite record to delete. The first index is one. Use htComprecCount to obtain the last valid index.

COUNT

The number of composite records to delete.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_COMPOSITE (invalid composite handle);

HTFAIL_END_OF_DATA (index is invalid).

Example

```
HTSTATUS status;
status = htComprecDelete (comphand, 10, 5);
```

See Also htComprecInsert, htComprecUpdate

htComprecFetch

Summary Retrieves the data from a composite record.

Syntax

```
HTSTATUS htComprecFetch (comphand, index, comptype,
                         compdata);
HTCOMPHANDLE   comphand;      /* Input */
HTINT               index;    /* Input */
HTCOMP         *comptype;     /* Input/Output, Optional
*/
void           *compdata;     /* Output */
```

Description Obtains the data from a specified composite record and copies it into a supplied structure. HiTest also copies any variable length component into a supplied buffer within the structure. Certain composite record types have no variable length component (e.g., PAB definition, paragraph). Other composite record types do have a variable length component (e.g., text, blob). Composite record data structures (HTC_ structures) with a pointer element have variable length components. Use htComprecLength to determine the length of the variable length component for a specific composite. The pointer element in the composite record structure must indicate a buffer (allocated by the calling program) of this length. To obtain a faster read-only, temporary direct pointer to the variable length component, use the similar function htComprecGetPtr.

Parameters

COMPHAND

The composite containing the composite record.

INDEX

The index of the composite record from which to obtain the data. The first index is one. Use htComprecCount to obtain the last valid index.

COMPTYPE

The type to fetch the composite record as. This value must be either HTCOMP_BLOB, the type of the composite record, or zero. If this value is zero, HiTest returns the composite record's type in this buffer. This type determines the format of the data at compdata.

COMPDATA

The data from the composite record. The data is one of multiple composite record structures (structure names prefixed with HTC_). The value of comptype determines the structure to use (i.e., if comptype is HTCOMP_TEXT, then compdata points to an HTC_TEXT structure). If the composite record structure has variable length data, allocate the buffer and set the pointer in the

composite record structure before calling this function. Use htComprecLength to determine the length of the buffer required.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_COMPOSITE (invalid composite handle);

HTFAIL_END_OF_DATA (index is invalid);

HTFAIL_OVERFLOW (record variable length component pointer is NULL);

HTFAIL_INVALID_CONVERT (composite records only convert to BLOB);

HTFAIL_DATA_UNAVAIL (cannot find doclink $LINKS item information);

HTFAIL_INVALID_DATABASE (cannot find doclink database);

HTFAIL_INVALID_DOCUMENT (cannot find doclink document);

HTFAIL_INVALID_VIEW (cannot find doclink view).

Example

```
HTC_PABDEF pabdef;
HTCOMP comptype = HTCOMP_PABDEF;
HTSTATUS status;
status = htComprecFetch (comphand, 1, &comptype, &pabdef);
```

See Also htComprecLength, htComprecGetPtr

htComprecGetPtr

Summary Retrieves the values and a data pointer from a composite record.

Syntax

```
HTSTATUS htComprecGetPtr (comphand, index, comptype,
                          compdata);
HTCOMPHANDLE   comphand;      /* Input */
HTINT               index;    /* Input */
HTCOMP         *comptype;     /* Input/Output, Optional
*/
void           *compdata;     /* Output */
```

Description Obtains the data from a specified composite record and copies it into a supplied structure. HiTest also returns a temporary, read-only pointer to any variable length component in the structure. Certain composite record types have no variable length component (e.g., PAB definition, paragraph). Other composite record types do have a variable length component (e.g., text, blob). Composite record data structures (HTC_ structures) with a pointer element have variable length components. To obtain a modifiable copy of the data contents, use the similar function htComprecFetch. This function is faster than htComprecFetch when accessing composite records with variable length data.

Parameters

COMPHAND

The composite containing the composite record.

INDEX

The index of the composite record from which to obtain the data. The first index is one. Use htComprecCount to obtain the last valid index.

COMPTYPE

The type to retrieve the composite record as. This value must be either HTCOMP_BLOB, the type of the composite record, or zero. If this value is zero, HiTest returns the composite record's type in this buffer. This type determines the format of the data at compdata.

COMPDATA

The data from the composite record. The data is one of multiple composite record structures (structure names prefixed with HTC_). The value of comptype determines the structure to use (i.e., if comptype is HTCOMP_TEXT, then compdata points to an HTC_TEXT structure). If the composite record structure has variable length data, then HiTest returns the pointer to this data in

the structure. This data should not be modified or freed by the calling program. The pointer becomes invalid at the next HiTest function call.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_COMPOSITE (invalid composite handle);

HTFAIL_END_OF_DATA (index is invalid);

HTFAIL_INVALID_CONVERT (composite records conversion not allowed);

HTFAIL_DATA_UNAVAIL (cannot find doclink $LINKS item information);

HTFAIL_INVALID_DATABASE (cannot find doclink database);

HTFAIL_INVALID_DOCUMENT (cannot find doclink document);

HTFAIL_INVALID_VIEW (cannot find doclink view).

Example

```
HTC_PABDEF pabdef;

HTCOMP comptype = HTCOMP_PABDEF;

HTSTATUS status;

status = htComprecGetPtr (comphand, 1, &comptype,
&pabdef);
```

See Also htComprecLength, htComprecFetch

htComprecInsert

Summary Inserts a single composite record into an existing composite.

Syntax

```
HTSTATUS htComprecInsert (comphand, index, comptype,
                          compdata);
HTCOMPHANDLE   comphand;          /* Input */
HTINT               index;        /*  Input,  Optional
*/
HTCOMP         comptype;          /* Input */
void           *compdata;         /* Input */
```

Description Inserts a new composite record into an existing composite at a specified location.

Parameters

COMPHAND

The composite into which to insert the composite record.

INDEX

The index at which to insert the composite record. The current composite record at this index will follow the new composite record. The first index is one. Use htComprecCount to obtain the last valid index. A value of zero directs HiTest to append this composite record to the end of the composite -- use index zero in multiple calls to append to a composite.

COMPTYPE

The composite record type of the new composite record. This type determines the format of the data at compdata.

COMPDATA

The data for the new composite record. The data is one of multiple composite record structures (structure names prefixed with HTC_). The value of comptype determines the structure to use (e.g., if comptype is HTCOMP_TEXT, then compdata points to an HTC_TEXT structure).

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_COMPOSITE (invalid composite handle);

HTFAIL_END_OF_DATA (index is invalid);

HTFAIL_BAD_FORMAT (cannot create composite record from compdata value);

HTFAIL_INVALID_CURSOR (doclink cursor is invalid);

HTFAIL_INVALID_DOCUMENT (doclink document handle is invalid);

HTFAIL_INVALID_VIEW (doclink view ID is invalid).

Example

```
HTSTATUS status;
HTC_PABREF pabref;
pabref.pabid = 2;
status = htComprecInsert (comphand, 10, HTCOMP_PABREF,
                          &pabref);
```

See Also htComprecUpdate, htComprecDelete

htComprecLength

Summary Obtains the length of the variable length portion of a composite record.

Syntax

```
HTSTATUS htComprecLength (comphand, index, comptype,
                          length);
HTCOMPHANDLE   comphand;          /* Input */
HTINT               index;       /* Input */
HTCOMP         *comptype;        /* Output */
HTINT               *length;     /* Output */
```

Description Obtains the length of the variable length portion of a specified composite record. Certain composite record types have no variable length component (e.g., PAB definition, paragraph). Other composite record types do have a variable length component (e.g., text, blob). Composite record data structures (HTC_ structures) with a pointer element have variable length components.

Parameters

COMPHAND

The composite containing the composite record.

INDEX

The index of the composite record whose length is to be determined. The first index is one. Use htComprecCount to obtain the last valid index.

COMPTYPE

The buffer to receive the composite record type of the specified composite record. This type determines the format of the data returned by a call to htComprecFetch or htComprecGetPtr.

LENGTH

The buffer to receive the length of the variable length component. If there is no variable length component, HiTest sets this value to zero.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_COMPOSITE (invalid composite handle);

HTFAIL_END_OF_DATA (index is invalid);

HTFAIL_INVALID_CONVERT (composite records only convert to BLOB).

Example

```
HTINT length;

HTCOMP comptype;

HTSTATUS status;

status = htComprecLength (comphand, 1, &comptype,
&length);
```

See Also htComprecFetch, htComprecGetPtr

htComprecList

Summary Iterates through composite records in a composite.

Syntax

```
HTSTATUS htComprecList (comphand, first, comptype, index);

HTCOMPHANDLE   comphand;      /* Input */

HTBOOL         first;    /* Input */

HTCOMP         *comptype;     /* Input/Output, Optional */

HTINT               *index;   /* Output */
```

Description Returns the first or next composite record information from the list of composite records in the composite. Use this function to list all composite records, or find those of a specific composite record type.

Parameters

COMPHAND

The composite from which to list composite records.

FIRST

Whether to get the first or next composite record. TRUE resets the composite record list, FALSE simply obtains the next composite record in the list. The value is always TRUE on the first call for a given composite handle.

COMPTYPE

Either the composite record type to find, or the buffer to receive the next composite record type. If nonzero, htComprecList finds the next composite record of this type. If zero, htComprecList finds the next composite record and returns its type in this location. To iterate through all composite records, remember to set this value to zero before each call.

INDEX

The buffer to receive the index of the composite record within the composite. Depending on the use of the comptype parameter, this will either be an incrementing index, or will jump (when finding records of a specific type). This index is useful as input to the various htComp and htComprec functions which require a composite record index.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_COMPOSITE (invalid composite handle);

HTFAIL_END_OF_DATA (no more composite records).

Example

```
HTCOMP comptype = 0;
HTINT index;
HTSTATUS status;
status = htComprecList (comphand, FALSE, &comptype, &index);
```

See Also htComprecCount

htComprecUpdate

Summary Modifies the contents of a single composite record.

Syntax

```
HTSTATUS htComprecUpdate (comphand, index, comptype,
                          compdata);
HTCOMPHANDLE   comphand;          /* Input */
HTINT               index;        /* Input */
HTCOMP         comptype;          /* Input */
void           *compdata;         /* Input */
```

Description Modifies the contents of a specified composite record. This is equivalent to deleting and then inserting a single composite record.

Parameters

COMPHAND

The composite containing the composite record to update.

INDEX

The index of the composite record to update. The first index is one. Use htComprecCount to obtain the last valid index.

COMPTYPE

The composite record type of the new composite record. This type determines the format of the data at compdata.

COMPDATA

The data for the new composite record. The data is one of multiple composite record structures (structure names prefixed with HTC_). The value of comptype determines the structure to use (e.g., if comptype is HTCOMP_TEXT, then compdata points to an HTC_TEXT structure).

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_COMPOSITE (invalid composite handle);

HTFAIL_END_OF_DATA (index is invalid);

HTFAIL_BAD_FORMAT (cannot create composite record from compdata value);

HTFAIL_INVALID_CURSOR (doclink cursor is invalid);

HTFAIL_INVALID_DOCUMENT (doclink document handle is invalid);

HTFAIL_INVALID_VIEW (doclink view ID is invalid).

Example

```
HTSTATUS status;
HTC_PABREF pabref;
pabref.pabid = 2;
status = htComprecUpdate (comphand, 10, HTCOMP_PABREF,
                          &pabref);
```

See Also htComprecInsert, htComprecDelete

Cursor

The cursor object has no context beneath the process level. Many of the remaining objects exist within the context of a cursor. Some objects use a cursor directly and some indirectly through a document, which is only valid in the context of its cursor. A cursor represents a session, which includes an open Lotus Notes database and state information. Each process or task can contain multiple cursors at any point in time. Multiple cursors may be open to a single database. All actions performed against a database occur through a cursor.

A cursor's state consists of an active form, an active view, an index, bindings, and open documents. Any operation which cancels or replaces part of a cursor's state destroys the previous value. For example, producing a new index destroys the previous index. Any state within the context of a cursor, including documents opened in the cursor, becomes invalid when closing the cursor. A cursor itself is a handle often used as a synonym for the session that it represents. HiTest uses the constant NULLHANDLE to represent an invalid cursor. The primary attributes of a cursor are the same as those of a database plus a handle.

The cursor group contains the following functions:

htCurClose	Closes an open cursor
htCurGetInfo	Obtains a piece of information from and about a cursor
htCurOpen	Opens a HiTest cursor
htCurReset	Resets the state of a cursor
htCurSetOption	Assigns the value of a cursor option

htCurClose

Summary Closes an open cursor.

Syntax

```
HTSTATUS htCurClose (cursor, force);
HTCURSOR  cursor;         /* Input */
HTBOOL    force;          /* Input */
```

Description Closes an open cursor. All data within the context of the cursor becomes invalid (e.g., all document and composite handles). This function commits changes to documents stored in this cursor with the bulk store option.

Parameters

CURSOR

The cursor to close.

FORCE

Indicates whether to force open documents in the cursor to close. If TRUE, HiTest closes and commits any open documents in the cursor with an internal call to htDocClose. If FALSE, any open documents cause the close to fail.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_OPEN_DOCUMENTS (cursor contains open documents and force is FALSE).

Example

```
HTSTATUS status;
status = htCurClose (cursor, TRUE);
```

See Also htCurOpen, htDocClose, htTerm

htCurGetInfo

Summary Obtains a piece of information from and about a cursor.

Syntax

```
HTSTATUS htCurGetInfo (cursor, item, buffer);
HTCURSOR  cursor;        /* Input */
HTCURINFO item;          /* Input */
void      *buffer;       /* Output */
```

Description Fetches one of various cursor-level information items into a supplied buffer. Each item has a data type, and the buffer must be large enough to hold the result.

Parameters

CURSOR

The cursor to use.

ITEM

One value from an enumeration of cursor items. Each item corresponds to a type (and length, for variable length types). The following table lists legal items with their corresponding data types and, where relevant, lengths:

constant	type
HTCURINFO_SERVERNAME	char [HTLEN_SERVERNAME + 1]
HTCURINFO_FILENAME	char [HTLEN_FILENAME + 1]
HTCURINFO_FORMNAME	char [HTLEN_DESIGNNAME + 1]
HTCURINFO_VIEWNAME	char [HTLEN_DESIGNNAME + 1]
HTCURINFO_DBTITLE	char [HTLEN_DATABASEINFO + 1]
HTCURINFO_DBCATEGORIES	char [HTLEN_DATABASEINFO + 1]
HTCURINFO_DBTEMPLATE	char [HTLEN_DATABASEINFO + 1]
HTCURINFO_DBDESIGNTEMPLATE	char [HTLEN_DATABASEINFO + 1]
HTCURINFO_DBHANDLE	standard Notes API: DBHANDLE

```
HTCURINFO_DBID          standard Notes API: DBID
```

SERVERNAME obtains the server name on which the database exists;

FILENAME obtains the database file name the cursor represents;

FORMNAME obtains the form name of the active form;

VIEWNAME obtains the view name of the active view;

DBTITLE obtains the database title;

DBCATEGORIES obtains the database categories;

DBTEMPLATE obtains the database template name;

DBDESIGNTEMPLATE obtains the database design template name;

The following items are only useful when combining HiTest calls with calls to the standard Notes API, and should be used carefully:

DBHANDLE obtains the standard Notes API database handle

DBID obtains the standard Notes API database ID

BUFFER

The buffer to receive the requested information. This buffer should be of sufficient length to handle the result.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_ILLEGAL_ENUM (invalid item).

Example

```
char dbtitle [HTLEN_DATABASEINFO + 1];
HTSTATUS status;
status = htCurGetInfo (cursor, HTCURINFO_DBTITLE,
dbtitle);
```

See Also htCurOpen

htCurOpen

Summary Opens a HiTest cursor.

Syntax

```
HTSTATUS htCurOpen (server, datapath, cursor);
char      *server;       /* Input, Optional */
char      *datapath;          /* Input */
HTCURSOR  *cursor;       /* Output */
```

Description Creates and returns a new cursor. The cursor represents a connection to the indicated database. Multiple cursors may be open at one time, including multiple cursors to a single database. All access to metadata and data within a database occurs through a cursor. Use the function htCurClose to close a cursor.

Parameters

SERVER

The server on which to open the database. To open a local database, use either NULL, the empty string, or the string assigned as the local server name with htSetOption.

DATAPATH

The file name and path relative to the Notes data directory of the database to open.

CURSOR

The buffer to receive the new cursor on successful operation.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_DATABASE (database does not exist).

Example

```
HTCURSOR cursor;
HTSTATUS status;
status = htCurOpen ("MyServer", "mail\JSmith.nsf",
&cursor);
```

See Also htSetOption, htCurClose

htCurReset

Summary Resets the state of a cursor.

Syntax

```
HTSTATUS htCurReset (HTCURSOR cursor);
HTCURSOR  cursor;         /* Input */
```

Description Clears the state of a cursor, as relates to the active index. This function performs the following actions:

1) clears the formula buffer,

2) clears the current (active) index,

3) clears all active bindings.

Parameters CURSOR

The cursor to reset.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR.

Example

```
HTSTATUS status;
status = htCurReset (cursor);
```

See Also htFormulaReset, htFormulaExec, htItemBind, htCellBind

htCurSetOption

Summary Assigns the value of a cursor option.

Syntax

```
HTSTATUS htSetOption (cursor, option, number, string);
HTCURSOR  cursor;        /* Input */
HTGLOBOPT option;        /* Input */
HTINT          number;       /* Input, Optional */
char           *string;      /* Input, Optional */
```

Description Assigns a value to the indicated cursor option. Depending on the option, the new value is supplied in either the number or string parameter.

Parameters

CURSOR

The cursor to use.

OPTION

One value from an enumeration of cursor options. Each option corresponds to either a string or integer value. This option indicates whether to use the string or numeric parameter (HiTest ignores the other parameter). The following table lists legal options with their corresponding data types, parameters, and defaults:

constant	type	parameter	default
HTCUROPT_BULK_STORE	HTBOOL	number	FALSE
HTCUROPT_STRICT_BIND	HTBOOL	number	TRUE
HTCUROPT_VIEW_POSITION	HTBOOL	number	FALSE
HTCUROPT_FETCH_SUMMARY	HTBOOL	number	FALSE
HTCUROPT_SUMMARY_LIMIT	HTINT	number	8192

See the htSetOption function for a description of the options.

NUMBER

The numeric or boolean value for an option. When setting the value of a boolean option, use the constants TRUE and FALSE.

STRING

The string value for an option.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_ILLEGAL_ENUM (invalid option);

HTFAIL_OVERFLOW (value out of bounds).

Example

```
HTSTATUS status;
status = htCurSetOption (cursor, HTGLOBOPT_BULK_STORE,
TRUE,
                         NULL)
```

See Also htCurOpen, htSetOption

Database

The database object has no context beneath the process level, but represents Lotus Notes databases as distinct objects. The primary attributes of a database are the server name, database name, and database file path (relative to the Notes data directory).

A database is a file used by Notes to store data in the form of documents. A database also contains metadata in the form of forms, views, and macros. A database can be accessed directly or through a Notes server. When accessed directly, Notes security is bypassed. When accessed through a Notes server, security is imposed, and the server properly handles multiple connections to a database.

The database group contains the following functions:

htDbGetPath	Obtains a database filename from a database title
htDbList	Iterates through available Notes databases and directories on a given server
htDbListCat	Iterates through databases in a database catalog

htDbGetPath

Summary Obtains a database filename from a database title.

Syntax

```
HTSTATUS htDbGetPath (server, directory, title,
catalog,
                      datapath);
char      *server;       /* Input, Optional */
char      *directory;    /* Input, Optional */
char      *title;        /* Input */
HTBOOL    catalog;       /* Input */
char      *datapath;          /* Output */
```

Description This function determines the filename of a database from the database's title by either performing a directory search or scanning the database catalog on a server. The calling program must supply the server name on which the database exists. This function is an inefficient way of finding databases (programs should try to retain database filepaths to avoid needing this function).

Parameters

SERVER

The server on which the database is located. To perform a local search, use either NULL, the empty string, or the string assigned as the local server name with htSetOption.

DIRECTORY

The directory to search for databases, relative to the Notes data directory. To search the data directory itself, use either NULL or the empty string. Catalog searches ignore this parameter (see catalog parameter).

TITLE

The title of the database to find. If two databases have the same title, this function will only find the first one.

CATALOG

Whether to perform a catalog search or a directory search. If TRUE, HiTest searches the database catalog (CATALOG.NSF) on the server for the title. If FALSE, HiTest performs a directory search in the directory indicated by the directory parameter.

DATAPATH

A character buffer which receives the database file name and path relative to the Notes data directory. The constant HTLEN_FILENAME defines the maximum datapath length.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_DATABASE (cannot find database).

Example

```
char dbfile [HTLEN_FILENAME + 1];
HTSTATUS status;
status = htDbGetPath ("MyServer", "mail", "JSmith.NSF",
                      FALSE, dbfile);
```

See Also htDbList, htDbListCat

htDbList

Summary Iterates through available Notes databases and directories on a given server.

Syntax

```
HTSTATUS htDbList (server, searchdir, operation,
recurse,
                   isdatabase, database, datafile);
char      *server;       /* Input, Optional */
char      *searchdir;    /* Input, Optional */
HTLIST    operation;          /* Input */
HTBOOL    recurse;       /* Input */
HTBOOL    *isdatabase;   /* Output, Optional */
char      *database;          /* Output, Optional */
char      *datafile;          /* Output, Optional */
```

Description Returns the first or next database information from the current database list. HiTest obtains the database list with a directory search depending on the input parameters. The database information available from this function consists of the database title and the database filename or filepath. In addition, this function returns directory entries. HiTest uses the values of the search parameters (server, searchdir, and recurse) to produce the database list.

Parameters

SERVER

The server from which to obtain the database list. To perform a local search, use either NULL, the empty string, or the string assigned as the local server name with htSetOption.

SEARCHDIR

The directory to search for databases, relative to the Notes data directory. To search the data directory itself, use either NULL or the empty string.

OPERATION

An element of the HTLIST enumeration that indicates whether and how to reset the database list. Use HTLIST_REFRESH to discard the database list and obtain a new list from Notes. If operation is HTLIST_FIRST and any of the search parameters are different from the values used in the previous search, then HiTest produces a new list using the new parameters. Then HiTest sets the next element in the list to the first element. Use HTLIST_NEXT to obtain the element following the last fetched element (HiTest ignores search parameters). The first call to this function following htInit always uses the value HTLIST_REFRESH.

RECURSE

Indicates whether to perform a recursive or flat search. A recursive search (TRUE) obtains all databases in or under the search directory. A nonrecursive search (FALSE) obtains databases and subdirectories in the search directory.

ISDATABASE

A boolean buffer set to TRUE if the current item is a database, and FALSE if the current item is a directory. For a recursive search, this value is always TRUE.

DATABASE

A character buffer which receives the database title. This will be the empty string for a directory. The constant HTLEN_DATABASEINFO defines the maximum title length.

DATAFILE

A character buffer which receives the database file. When performing a nonrecursive search, this buffer receives the filename only, and not the path. When performing a recursive search, this buffer receives the full file name and path relative to the Notes data directory. The constant HTLEN_FILENAME defines the maximum datafile length.

Returns HTSTATUS return code. Failures include:

HTFAIL_END_OF_DATA (no more results).

Example

```
char dbtitle [HTLEN_DATABASEINFO + 1];
char dbfile [HTLEN_FILENAME + 1];
HTBOOL isdb;
HTLIST list_op = HTLIST_FIRST;
printf ("List of Local Notes Databases:");
while (!htDbList (NULL, NULL, list_op, FALSE, &isdb,
                dbtitle, dbfile))
{
     if (!isdb)
          continue;
     printf ("\n       Title: '%s',   Filename: '%s' ",
dbtitle,
          dbfile);
```

```
        list_op = HTLIST_NEXT;
    }
```

See Also htDbListCat, htSetOption

htDbListCat

Summary Iterates through databases in a database catalog.

Syntax

```
HTSTATUS htDbListCat (server, operation, database,
                      datapath);
char      *server;       /* Input, Optional */
HTLIST    operation;          /* Input */
char      *database;          /* Output, Optional */
char      *datapath;          /* Output, Optional */
```

Description Returns the first or next database information for the databases in a database catalog. A database catalog must be available on the selected Notes server.

Parameters

SERVER

The server on which to examine the database catalog. To use the local database catalog, use either NULL, the empty string, or the string assigned as the local server name with htSetOption. There must be a database catalog CATALOG.NSF in the Notes data directory on the server.

OPERATION

An element of the HTLIST enumeration that indicates whether and how to reset the database list. Use HTLIST_REFRESH to discard the database list and obtain a new list from the database catalog. If operation is HTLIST_FIRST and the server name is different from the value used in the previous search, then HiTest uses the new server's catalog. Then HiTest sets the next element in the list to the first element. Use HTLIST_NEXT to obtain the element following the last fetched element (HiTest ignores search parameters). The first call to this function following htInit always uses the value HTLIST_REFRESH.

DATABASE

A character buffer which receives the database title. HiTest returns the empty string for a directory. The constant HTLEN_DATABASEINFO defines the maximum title length.

DATAPATH

A character buffer which receives the database file name and path relative to the Notes data directory. The constant HTLEN_FILENAME defines the maximum datapath length

Returns HTSTATUS return code. Failures include:

HTFAIL_DATA_UNAVAIL (requested server catalog is unavailable);

HTFAIL_END_OF_DATA (no more catalog entries).

Example

```
char dbtitle [HTLEN_DATABASEINFO + 1];
char dbfile [HTLEN_FILENAME + 1];
HTLIST list_op = HTLIST_FIRST;
printf ("List of Databases in Server 'MyServer' Catalog:");
while (!htDbListCat ("MyServer", list_op, dbtitle, dbfile))
{
     if (isdb)
          printf ("\n       Title: '%s',   Filename: '%s' ",
               dbtitle, dbfile);
     list_op = HTLIST_NEXT;
}
```

See Also htDbList, htDbGetPath

Datetime

Use Datetime functions to access and manipulate the components of datetime items. These functions allow simple control of datetime data. The internal structure of a datetime is not accessible to API programs.

The following structure represents a datetime, but should not be manipulated directly

```
typedef struct
{
   DWORD innards [2];
} HTDATETIME;       /* Datetime structure */
```

Use the following structure to create, access, and update datetimes on a by-component basis

```
typedef struct
{
   short int year;        /* Year (1-32767) */
   short int month;  /* Month (1-12) */
   short int dom;         /* Day of month (1-31) */
   short int weekday;     /* Day of week (1-7 where Sunday = 1) */
   short int hour;        /* Hour (0-23) */
   short int minute; /* Minute (0-59) */
   short int second; /* Second (0-59) */
   short int hundsec;     /* Hundredths of second (0-99) */
   HTBOOL dst;       /* Whether daylight savings is in effect */
   short int zone;        /* Time zone (-11 to 11) */
} HTTIMESUMM;       /* Datetime component structure */
```

The datetime group contains the following functions:

htDatetimeCompare	Relatively compares two datetimes
htDatetimeCreate	Creates a datetime from individual components
htDatetimeDiff	Absolutely compares two datetimes
htDatetimeGetInfo	Obtains a piece of information from and about a datetime
htDatetimeUpdate	Modifies a datetime to produce a new datetime

htDatetimeCompare

Summary Relatively compares two datetimes.

Syntax

```
HTINT htDatetimeCompare (datetime1, datetime2);
HTDATETIME          *datetime1;          /* Input */
HTDATETIME          *datetime2;          /* Input */
```

Description Compares two datetimes and returns whether the first is greater than, equal to, or less than the second. To determine the absolute difference in time between two datetimes, use the htDatetimeDiff function.

Parameters

DATETIME1

A pointer to the first datetime.

DATETIME2

A pointer to the second datetime.

Returns HTINT relation of the first and second datetimes. If the first datetime is greater than the second, returns greater than zero. If the first datetime is less than the second, returns less than zero. If the two datetimes are equal, returns zero.

Example

```
HTINT compare;
compare = htDatetimeCompare (&datetime1, &datetime2);
```

See Also htDatetimeGetInfo, htDatetimeDiff

htDatetimeCreate

Summary Creates a datetime from individual components.

Syntax

```
HTSTATUS htDatetimeCreate (timesumm, autozone, datetime);
HTTIMESUMM          *timesumm;          /* Input */
HTBOOL         autozone;      /* Input */
HTDATETIME          *datetime;          /* Output */
```

Description Creates a datetime from individual date and time components. The time zone and daylight savings time status may be provided as input or determined automatically from the Notes installation.

Parameters

TIMESUMM

A pointer to the structure containing the datetime components. This function ignores the weekday field.

AUTOZONE

Whether to determine automatically the time zone and daylight savings time status from the Notes installation. A value of TRUE ignores these components in the timesumm structure and automatically determines them. A value of FALSE uses the values in the timesumm structure.

DATETIME

The buffer to receive the new datetime value.

Returns HTSTATUS return code. Failures include:

HTFAIL_BAD_FORMAT (the timesumm values do not represent a valid datetime).

Example

```
HTDATETIME datetime;
HTTIMESUMM timesumm;
HTSTATUS status;
timesumm.year = 1999;
timesumm.month = 12;
timesumm.dom = 31;
```

```
timesumm.hour = 23;
timesumm.minute = 59;
timesumm.second = 59;
timesumm.hundsec = 99;
status = htDatetimeCreate (&timesumm, TRUE, &datetime);
```

See Also htDatetimeGetInfo, htDatetimeUpdate

htDatetimeDiff

Summary Absolutely compares two datetimes.

Syntax

```
HTINT htDatetimeDiff (datetime1, datetime2);
HTDATETIME          *datetime1;          /* Input */
HTDATETIME          *datetime2;          /* Input */
```

Description Compares two datetimes and returns the number of seconds difference between the two. To simply determine which datetime is greater, use the faster htDatetimeCompare function.

Parameters

DATETIME1

A pointer to the first datetime.

DATETIME2

A pointer to the second datetime.

Returns HTINT number of seconds between the first and second datetimes. If the first datetime is greater than the second, the value is positive. If the first datetime is less than the second, the value is negative. If the two datetimes are equal, returns zero.

Example

```
HTINT seconds;
seconds = htDatetimeDiff (&datetime1, &datetime2);
```

See Also htDatetimeGetInfo, htDatetimeCompare

htDatetimeGetInfo

Summary Obtains a piece of information from and about a datetime.

Syntax

```
HTSTATUS htDatetimeGetInfo (datetime, item, buffer);
HTDATETIME     datetime;           /* Input */
HTTIMEINFO     item;          /* Input */
void      *buffer;       /* Output */
```

Description Fetches one of various datetime-level information items into a supplied buffer. Each item has a data type, and the buffer must be large enough to hold the result.

Parameters

DATETIME

The datetime to use.

ITEM

One value from an enumeration of datetime items. Each item corresponds to a type (and length, for variable length types). The following table lists legal items with their corresponding data types and, where relevant, lengths:

constant	type
HTTIMEINFO_JULIAN	HTINT
HTTIMEINFO_TICKS	HTINT
HTTIMEINFO_HTTIMESUMM	HTTIMESUMM

JULIAN obtains the julian date from the date component of the datetime;

TICKS obtains the number of ticks (hundredths of a second) since midnight;

HTTIMESUMM obtains the datetime components.

BUFFER

The buffer to receive the requested information. This buffer should be of sufficient length to handle the result.

Returns HTSTATUS return code. Failures include:

HTFAIL_BAD_FORMAT (invalid datetime);

HTFAIL_ILLEGAL_ENUM (invalid item).

Example

```
HTINT julian;
HTSTATUS status;
status = htDatetimeGetInfo (&datetime,
HTTIMEINFO_JULIAN,
                            &julian);
```

See Also htDatetimeCreate

htDatetimeUpdate

Summary Modifies a datetime to produce a new datetime.

Syntax

```
HTSTATUS htDatetimeUpdate (timesumm, datetime);
HTTIMESUMM          *timesumm;          /* Input */
HTDATETIME          *datetime;          /*       Input, Output */
```

Description Modifies the components of a datetime to produce a new datetime.

Parameters

TIMESUMM

A pointer to the structure containing the modifications. HiTest adds the value of each component to the datetime (e.g., a month field of two increases the month of the datetime by two). This function ignores the weekday, time zone, and daylight savings time status.

DATETIME

A pointer to the datetime to update.

Returns HTSTATUS return code. Failures include:

HTFAIL_OVERFLOW (the update would result in an invalid datetime).

Example

```
HTTIMESUMM timesumm;
HTSTATUS status;
timesumm.year = 1;
timesumm.month = 1;
timesumm.dom = 1;
timesumm.hour = 1;
timesumm.minute = 1;
timesumm.second = 1;
timesumm.hundsec = 1;
status = htDatetimeUpdate (&timesumm, &datetime);
```

See Also htDatetimeCreate, htDatetimeGetInfo

Document

Documents are the primary components of databases. Each document in turn contains items. Each document has a document ID which is unique within the document's database. Use this ID to reference individual documents. Sets of documents may be obtained from an index, which is produced by executing a formula. The primary attribute of a document is the document ID.

To access or manipulate the contents of a document, the document must be opened. HiTest represents an open document by a document handle. A document handle is valid until closing either the document itself or the cursor containing the open document. HiTest uses the constant NULLHANDLE to represent an invalid document handle.

The document group contains the following functions:

htDocClose	Closes an open document, optionally discarding all changes
htDocCopy	Copies a document, and optionally its hierarchy, between cursors
htDocCreate	Creates a new, empty document
htDocDelete	Deletes a document, and optionally its hierarchy
htDocFetch	Opens the next document in the active index, and loads all bound data
htDocGetInfo	Obtains a piece of information from and about an open document
htDocOpen	Opens a document for data access or modification
htDocPut	Creates a new document from bound items
htDocUpdate	Updates modifications to bound items back to the last fetched document

htDocClose

Summary Closes an open document, optionally discarding all changes.

Syntax

```
HTSTATUS htDocClose (dochand, commit);
HTDOCHANDLE          dochand;        /* Input */
HTBOOL          commit;         /* Input */
```

Description Closes a document, invalidating the handle. This function also controls whether to save or discard changes made to the document. Documents not closed with this function are eventually closed by either htCurClose or htTerm, both of which automatically commit changes.

Parameters

DOCHAND

Handle of the document to close.

COMMIT

Whether to save changes. If FALSE, HiTest discards all changes to the document. If TRUE and there are changes to document items, then HiTest saves those changes into the database.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_DOCUMENT (invalid document handle);

Example

```
htDocClose (dochand, TRUE);
```

See Also htDocOpen, htDocCreate, htCurClose, htTerm

htDocCopy

Summary Copies a document, and optionally its hierarchy, between cursors.

Syntax

```
HTSTATUS htDocCopy (src_cursor, src_docid, dest_cursor,
                    view_docid, dest_docid);
HTCURSOR  src_cursor;    /* Input */
HTDOCID   src_docid;          /* Input */
HTCURSOR  dest_cursor;   /* Input */
HTVIEWID  viewid;        /* Input, Optional */
HTDOCID   *dest_docid;   /* Output, Optional */
```

Description Creates a copy of the source document in the destination cursor. This function optionally also copies the source document's response hierarchy into the destination cursor.

Parameters

SRC_CURSOR

The cursor containing the source document.

SRC_DOCID

The document to copy.

DEST_CURSOR

The cursor into which to copy the document.

VIEWID

This parameter determines whether to perform a hierarchy copy. Use NULLID to copy the source document only. To copy a document and its entire response hierarchy, use the HTVIEWID of the view containing the document and the hierarchy. When copying a single document, using a view reduces performance.

DEST_DOCID

The buffer to receive the document ID for the new copy of the source document.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_INVALID_DOCUMENT (document does not exist);

HTFAIL_INVALID_VIEW (view does not exist).

Example

```
HTDOCID new_docid;

HTSTATUS status;

status = htDocCopy (cursor1, old_docid, cursor2,
viewid,
                    &new_docid);
```

See Also htDocDelete, htViewGetId

htDocCreate

Summary Creates a new, empty document.

Syntax

```
HTSTATUS htDocCreate (cursor, formname, dochand);
HTCURSOR       cursor;        /* Input */
char           *formname;          /*  Input,  Optional
*/
HTDOCHANDLE         *dochand;           /* Output */
```

Description Creates and opens a new document in the cursor. Once created, the document behaves like any other document. If bulk storage is active, then this document will not have a valid document ID until the containing cursor is closed.

Parameters

CURSOR

The cursor in which to create the new document.

FORMNAME

The form to use for the new document. A form is required when strict binding is in effect.

DOCHAND

The buffer to receive the new document handle. Use htDocClose to close the new document.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_FORM_UNAVAIL (strict binding requires an active form).

Example

```
DOCHANDLE dochand;
HTSTATUS status;
status = htDocCreate (cursor, "Person", &dochand);
```

See Also htDocOpen, htDocClose

htDocDelete

Summary Deletes a document, and optionally its hierarchy.

Syntax

```
HTSTATUS htDocDelete (cursor, docid, viewid);
HTCURSOR  cursor;         /* Input */
HTDOCID   docid;          /* Input */
HTVIEWID  viewid;         /* Input, Optional */
```

Description Deletes the indicated document. This function optionally also deletes the document's response hierarchy.

Parameters

CURSOR

The cursor containing the document.

DOCID

The document to delete.

VIEWID

This parameter determines whether to perform a hierarchy delete. Use NULLID to delete the indicated document only. To delete a document and its entire response hierarchy, use the HTVIEWID of the view containing the document and the hierarchy. When copying a single document, using a view reduces performance.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_INVALID_DOCUMENT (document does not exist);

HTFAIL_INVALID_VIEW (view does not exist).

Example

```
HTSTATUS status;
status = htDocDelete (cursor, docid, viewid);
```

See Also htViewGetId, htDocCopy

htDocFetch

Summary Opens the next document in the active index, and loads all bound data.

Syntax

```
HTSTATUS htDocFetch (cursor, dochand);
HTCURSOR       cursor;        /* Input */
HTDOCHANDLE         *dochand;           /*      Output,
Optional */
```

Description Loads all bound data from the next document in the active index. Bound data consists of all items and cells bound since the last htFormulaExec. Use this function when loading the same data from multiple documents in an index. This function performs a document open, data access of multiple items and cells, and automatic conversion as one operation. The document remains open as long as it is the current document in the active index. During this time, the document handle returned from this function is a valid document handle. HiTest closes the document handle when navigating within the index or destroying the index.

If the cursor option FETCH_SUMMARY is active, only summary document items are available. This is more efficient than loading non-summary data, but prevents access to composite items and some other large items. The global option TEXT_TRUNCATE causes automatic truncation of data retrieved as text to fit in buffers bound with insufficient lengths. Other retrievals (to types other than text, or to text when TEXT_TRUNCATE is inactive) do not allow truncation. If there are no bound items (i.e., only cells), then the document is not opened, and the document handle is unavailable.

Use this function with htDocUpdate to allow easy modification and updating of multiple bound items.

Parameters

CURSOR

The cursor containing the relevant index.

DOCHAND

The buffer to receive the fetched document's handle. When fetching a document, the document remains opened as long as it is the current entry in the index (i.e., until index navigation or destruction). HiTest closes the document (and reuses the document handle) when it is no longer the current index entry. If there are no bound items, the document is not opened (this buffer receives the value NULLHANDLE).

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_DATA_UNAVAIL (no active index);

HTFAIL_OVERFLOW (data does not fit in bound buffer).

Example

```
HTDOCHANDLE dochand;
HTSTATUS status;
status = htDocFetch (cursor, &dochand);
```

See Also htSetOption, htCurReset, htFormulaExec, htIndexNavigate, htItemBind, htCellBind, htDocUpdate

htDocGetInfo

Summary Obtains a piece of information from and about an open document.

Syntax

```
HTSTATUS htDocGetInfo (dochand, item, buffer)
HTDOCHANDLE         dochand;       /* Input */
HTGLOBINFO          item;          /* Input */
void           *buffer;       /* Output */
```

Description Fetches one of various document-level information items into a supplied buffer. Each item has a data type, and the buffer must be sufficiently large to hold the result.

Parameters

DOCHAND

The document on which to obtain information.

ITEM

One value from an enumeration of document items. Each item corresponds to a type (and length, for variable length types). The following table lists legal items with their corresponding data types and, where relevant, lengths:

constant	type
HTDOCINFO_CREATED	HTDATETIME
HTDOCINFO_LASTMODIFIED	HTDATETIME
HTDOCINFO_FORMNAME	char [HTLEN_DESIGNNAME + 1]
HTDOCINFO_TITLELENGTH	HTINT
HTDOCINFO_TITLESTRING	char [HTDOCINFO_TITLELENGTH + 1]
HTDOCINFO_ISDIRTY	HTBOOL
HTDOCINFO_HTDOCID	HTDOCID
HTDOCINFO_HTCURSOR	HTCURSOR
HTDOCINFO_FILECOUNT	HTINT
HTDOCINFO_NOTEID	Standard Notes API: NOTEID
HTDOCINFO_NOTEHANDLE	Standard Notes API: NOTEHANDLE

CREATED obtains the document's original creation time;

LASTMODIFIED obtains the document's last modified time;

FORMNAME obtains the document's form name, if any;

TITLELENGTH obtains the length of the document window title;

TITLESTRING obtains the document window title;

ISDIRTY obtains a boolean which indicates whether data has been altered;

HTDOCID obtains the HiTest document ID;

HTCURSOR obtains the HiTest cursor in which this document was opened;

FILECOUNT obtains the number of file attachments in the document.

Use the following items carefully, since they are only useful when integrating HiTest calls with calls to the standard Notes API:

NOTEID obtains the standard Notes API note ID;

NOTEHANDLE obtains the standard Notes API note handle.

The document window title has some special properties. To provide a buffer of sufficient size, predetermine the length by calling the function htDocGetInfo with the HTDOCINFO_TITLELENGTH item. Additionally, some window titles may contain view-specific results (e.g., number of responses). If this document is the current document in the active view-based index, then the title includes the view information (otherwise, this function deletes view-specific information from the title string).

BUFFER

The buffer to receive the requested information. This buffer should be of sufficient length to handle the result.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_DOCUMENT (invalid document handle);

HTFAIL_ILLEGAL_ENUM (invalid item).

Example

```
HTDATETIME last_mod;
HTSTATUS status;
status = htDocGetInfo (dochand, HTDOCINFO_LASTMODIFIED,
                       &last_mod);
```

See Also htDocFetch, htDocOpen, htDocCreate

htDocOpen

Summary Opens a document for data access or modification.

Syntax

```
HTSTATUS htDocOpen (cursor, docid, objects, dochand);
HTCURSOR        cursor;         /* Input */
HTDOCID         docid;          /* Input */
HTBOOL          summary;        /* Input */
HTDOCHANDLE          *dochand;           /* Output */
```

Description Opens a document and returns a handle. Use this handle in other HiTest functions to obtain and modify data in the document. Close the document with htDocClose to store or discard changes to the document data. The cursor's active form has no relevance for documents opened with this function -- the document uses its own form.

Use this method of data access when handling single or multiple documents inconsistently. Use htDocFetch when performing the same actions on multiple documents in an index.

Parameters

CURSOR

The cursor containing the document.

DOCID

The document to open.

SUMMARY

Whether to load only summary items. Composite items are never summary items, and other items usually are (this is not always true, since a document may contain no more than 15K of summary data). If this parameter is TRUE, then only summary items will be available from this document. If FALSE, then all document items will be available. Do not load non-summary objects unless required, since they are often large.

DOCHAND

The buffer to receive the new document handle. This handle is valid until the document or the containing cursor is closed.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_INVALID_DOCUMENT (document does not exist).

Example

```
HTDOCHANDLE dochand;
HTSTATUS status;
status = htDocOpen (cursor, docid, TRUE, &dochand);
```

See Also htDocClose, htDocFetch, htDocGetInfo, htIndexNavigate, htCurClose

htDocPut

Summary Creates a new document from bound items.

Syntax

```
HTSTATUS htDocPut (cursor, docid);
HTCURSOR  cursor;         /* Input */
HTDOCID   *docid;         /* Output, Optional */
```

Description Creates a new document and new items in that document from bound items. All bound items (not bound cells) are converted and stored in the new document. If strict binding is in effect, then the new document uses the active form. If this is the first call to htDocPut since a call to htFormTemplate, then this document uses a form template from htFormTemplate. In this case, form creation of the virtual form precedes the new document's creation. The created form has the same fields as the document's items.

Parameters

CURSOR

The cursor in which to create the document.

DOCID

The buffer to receive the new document's ID. If bulk storage is active, then documents created with htDocPut do not have a valid document ID until the containing cursor is closed, and this value will be NULLID. Otherwise, this document ID behaves like any normal document.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_FORM_UNAVAIL (strict binding requires an active form).

Example

```
HTDOCID docid;
HTSTATUS status;
status = htDocPut (cursor, &docid);
```

See Also htCurReset, htItemBind

htDocUpdate

Summary Updates modifications to bound items back to the last fetched document.

Syntax

```
HTSTATUS htDocUpdate (cursor);
HTCURSOR  cursor;         /* Input */
```

Description Updates certain bound items back to the previously fetched document. This function is only valid following a htDocFetch, and while the fetched document is still the current document in the active index. This function copies into the document any modifications to bound items in memory. Modifications include not only the item data, but also any bound length and null indicators). This function only updates an item if the item's bound update boolean assigned with htItemBind is set to TRUE.

Parameters CURSOR

The cursor containing the index.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_DATA_UNAVAIL (no open fetched document from an active index).

Example

```
HTSTATUS status;
status = htDocUpdate (cursor);
```

See Also htIndexNavigate, htDocFetch, htItemBind, htCurReset

Error

Each HiTest process or task contains error information accessible in various ways. Since different programmers prefer different methods, HiTest has three error handling methods: simple error retrieval, error writeback buffers, and an error callback procedure. Each process or task has its own error information. Calling a HiTest function will clear any error information from a previous function call. This does not apply to the htError functions, which have no effect on the current error information.

See the “Error Handling” section of Chapter 3, “Programming to the HiTest API” for a discussion of error handling and a list of error codes and severities.

The error group contains the following functions:

htErrorFetch	Obtains the current error information
htErrorSetBuffer	Sets writeback buffers to receive error information on any HiTest error
htErrorSetProc	Assigns an error callback function

htErrorFetch

Summary Obtains the current error information.

Syntax

```
HTINT htErrorFetch (length, status, severity, message);
HTINT          length;        /* Input, Optional */
HTSTATUS  *status;       /* Output, Optional */
HTSEVERITY     *severity;          /* Output, Optional
*/
char      *message;      /* Output, Optional */
```

Description Obtains the information for any error produced by the most recent HiTest function call. This information consists of an error status code, an error severity, and an error message. When an error occurs, the error information is accessible until the next HiTest call (excluding htError functions), at which time HiTest resets the error information.

Parameters

LENGTH

The length of the supplied error message buffer in the message parameter. Use a value of zero if the buffer is of sufficient size to hold the message. The constant HTLEN_ERROR defines the maximum error message length.

STATUS

The buffer to receive the error status code.

SEVERITY

The buffer to receive the severity of the error code.

MESSAGE

The buffer to receive the error message string. Supply the buffer length in the length parameter.

Returns HTINT error string length. Returns zero if there is no current error or the message parameter is NULL.

Example

```
char message [HTLEN_ERROR + 1];
HTINT length;
```

```
HTSTATUS status;
HTSEVERITY severity;
length = htErrorFetch (0, &status, &severity,
&message);
```

See Also htErrorSetBuffer, htErrorSetProc

htErrorSetBuffer

Summary Sets writeback buffers to receive error information on any HiTest error.

Syntax

```
void htErrorSetBuffer (length, buffer, status, severity);

HTINT          length;        /* Input, Optional */

char      *buffer;       /* Input, Optional */

HTSTATUS  *status;       /* Input, Optional */

HTSEVERITY     *severity;          /* Input, Optional */
```

Description Sets writeback buffers which will automatically receive the error string, error status code, and error severity when a HiTest function generates an error. If there is an assigned error callback function (by htErrorSetProc), then HiTest writes the error information to the writeback buffers before calling the callback function. Normally, using both writeback buffers and a callback function is unnecessary.

Parameters

LENGTH

The length of the buffer which will receive the error message. The constant HTLEN_ERROR defines the maximum error message length. A value of zero indicates that the buffer is of sufficient length to contain the result.

BUFFER

The buffer to receive the error message string.

STATUS

The buffer to receive the error code.

SEVERITY

The buffer to receive the error severity.

Returns void.

Example

```
char error [HTLEN_ERROR + 1];

HTSTATUS status;
```

```
HTSEVERITY severity;

htErrorSetBuffer    (HTLEN_ERROR,    error,    &status,
&severity);
```

See Also htErrorFetch, htErrorSetProc

htErrorSetProc

Summary Assigns an error callback function.

Syntax

```
HTERRORPROC htErrorSetProc (errproc, errparam);
HTERRORPROC          errproc;        /* Input */
void            *errparam;           /*  Input,  Optional
*/
```

Description Assigns an error callback function. When a HiTest function generates an error, HiTest calls the callback function before the HiTest function returns. If there are assigned error writeback buffers (by htErrorSetBuffer), then HiTest writes the error information to the writeback buffers before calling the callback function. Normally, using both writeback buffers and a callback function is unnecessary.

Parameters ERRPROC

The address of the callback function. Define the callback function based on the HTERRORPROC prototype declaration:

```
typedef HTSTATUS (far HTAPIERRTYPE HTERRORPROC)
                                  (HTSTATUS code,
                                   HTSEVERITY severity,
                                   char far *errmsg,
                                   void far *buffer);
```

The callback function parameters are:

code is the error status code;

severity is the error severity;

errmsg is a read-only pointer to the error message string;

buffer is the errparam parameter supplied to htErrorSetProc.

The constant HTAPIERR defines the platform independent calling convention for error callback functions.

ERRPARAM

A parameter supplied when HiTest calls the callback function. This provides a method for the calling program to transmit its own context information to the callback function.

Returns HTERRORPROC pointer to the callback function prior to this function call. Returns NULL if there was no previous callback function.

Example

```
HTSTATUS HTAPIERR HiTestErrorProc (HTSTATUS code,
                                   HTSEVERITY severity,
                                   char far *errmsg,
                                   void far *buffer);


HTERRORPROC old_errorproc;
old_errorproc = htErrorSetProc (HiTestErrorProc,
context);
```

See Also htErrorFetch, htErrorSetBuffer

Field

A field is the component of a form which describes a single data item within a document. Each form contains one or more fields. The primary attributes of a field are a name and data type. When strict binding is in effect, all items within a document must have a corresponding field within the document's form.

The following flags define field attributes in the HTFIELD structure:

HTFIELD_READWRITERS	Field contains readwriter names
HTFIELD_EDITABLE	Field may be edited
HTFIELD_NAMES	Field contains distinguished names
HTFIELD_STOREDV	Always store default values
HTFIELD_READERS	Field contains document readers
HTFIELD_SECTION	Field contains a section
HTFIELD_COMPUTED	Computed field
HTFIELD_KEYWORDS	Keywords field
HTFIELD_PROTECTED	Field is protected
HTFIELD_REFERENCE	Name is reference to a shared field
HTFIELD_SIGN	Field is signed
HTFIELD_SEAL	Field is sealed (encrypted)
HTFIELD_KWD_UI_STD	Keywords UI is standard
HTFIELD_KWD_UI_CHECK	Keywords UI is a checkbox
HTFIELD_KWD_UI_RADIO	Keywords UI is a radio button
HTFIELD_KWD_UI_NEW	Allow new keywords

htFieldList and htFieldGetInfo return the following field attribute structure:

```
typedef struct
{
   HTTYPE type;                 /* Field data type */
   char name [HTLEN_FIELDNAME + 1]; /* Field name */
   HTFLAGS flags;                 /*  Field  flags  (HTFIELD_xxx)
*/
   HTINT desc_len;                 /* Length of the field
                                 description */
   HTINT keylist_len;              /* Length of the keywords text
                                 list */
} HTFIELD;                         /*  Field  attribute  structure
*/
```

The field group contains the following functions:

htFieldCount	Obtains the number of fields in a form
htFieldGetInfo	Obtains a piece of information about a form field
htFieldList	Iterates through fields in a form

htFieldCount

Summary Obtains the number of fields in a form.

Syntax

```
HTSTATUS htFieldCount (cursor, formid, fieldcount);
HTCURSOR  cursor;        /* Input */
HTFORMID  formid;        /* Input */
HTINT          *fieldcount;   /* Output */
```

Description Obtains the number of fields in the indicated form.

Parameters

CURSOR

The cursor containing the form.

FORMID

The form from which to obtain the field count.

FIELDCOUNT

The buffer to receive the number of fields in the form.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_INVALID_FORM (form does not exist).

Example

```
HTINT fieldcount;
HTSTATUS status;
status = htFieldCount (cursor, formid, fieldcount);
```

See Also htFormGetId, htFieldList

htFieldGetInfo

Summary Obtains a piece of information about a form field.

Syntax

```
HTSTATUS htFieldGetInfo (cursor, formid, fieldname,
item,
                         buffer);
HTCURSOR        cursor;         /* Input */
HTFORMID        formid;         /* Input */
char            *fieldname;     /* Input */
HTFIELDINFO          item;           /* Input */
void            *buffer;        /* Output */
```

Description Fetches one of various field information items into a supplied buffer. Each item has a data type, and the buffer must be sufficiently large to hold the result. To obtain either the field description or field keywords list, determine the required buffer length from the field's HTFIELD structure. Lengths of zero indicate that the information is not used for the field. HiTest returns keywords as a text list (see the htTextList functions).

Parameters

CURSOR

The cursor containing the form.

FORMID

The form from which to obtain the field information.

FIELDNAME

The name of the desired field within the form.

ITEM

One value from an enumeration of field items. Each item corresponds to a type (and length, for variable length types). The following table lists legal items with their corresponding data types and, where relevant, lengths:

constant	type
HTFIELDINFO_HTFIELD	HTFIELD *

```
HTFIELDINFO_DESCRIPTION        char [HTFIELD.desc_len]
HTFIELDINFO_KEYWORDS           HTTYPE_TEXT_LIST,
                            length: HTFIELD.keylist_len
```

HTFIELD obtains the HTFIELD information structure.

DESCRIPTION obtains the field description;

KEYWORDS obtains the field keywords text list. Each element in this text list is one keyword. Each keyword may contain one or more synonyms in the form of "KEY1|KEY2", where the leftmost string is the value displayed in the UI, and the rightmost string is the value stored in the item;

BUFFER

The buffer to receive the requested information. This buffer should be of sufficient length to handle the result.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_INVALID_FORM (form does not exist);

HTFAIL_INVALID_FIELD (no such field in the form);

HTFAIL_ILLEGAL_ENUM (invalid item).

Example

```
HTFIELD htfield;
HTSTATUS status;
status = htFieldGetInfo (HTFIELDINFO_HTFIELD, &field);
```

See Also htFormGetId, htFieldList

htFieldList

Summary Iterates through fields in a form.

Syntax

```
HTSTATUS htFieldList (cursor, formid, first, field);
HTCURSOR  cursor;        /* Input */
HTFORMID  formid;        /* Input */
HTBOOL    first;         /* Input */
HTFIELD   *field;        /* Output */
```

Description Returns the first or next field information from the list of fields in the form.

Parameters

CURSOR

The cursor containing the form.

FORMID

The form from which to list fields.

FIRST

Whether to get the first or next field. TRUE resets the field list, FALSE obtains the next field in the list. This value is always TRUE on the first call for a given form.

FIELD

The structure to receive information on the field. See the Field object section preceding the htField functions for a description of this structure and its contents. The description and keyword list lengths define the required buffer length for htFieldGetInfo when obtaining the field description and keywords, respectively.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_INVALID_FORM (form does not exist);

HTFAIL_END_OF_DATA (no more fields).

Example

```
HTFIELD field;
```

```
HTSTATUS status;
status = htFieldList (cursor, FALSE, &field);
```

See Also htFormGetId, htFieldCount, htFieldGetInfo

File

Internally, Lotus Notes stores file attachments as items of a particular data type. The HiTest API handles files differently since they may have different actions performed on them, specifically attaching and extracting. Notes stores files attached to a document within items named “$FILE”, and these are the items accessed by the file functions. The primary attribute of a file is a file name (which does not include any path information).

The file group contains the following functions:

htFileDelete	Deletes a file attachment from a document
htFileFetch	Extracts a file attachment from a document to a file
htFileList	Iterates through file attachments in a document
htFilePut	Attaches a file to a document

htFileDelete

Summary Deletes a file attachment from a document.

Syntax

```
HTSTATUS htFileDelete (dochand, filename);
HTDOCHANDLE         dochand;       /* Input */
char           *filename;          /* Input */
```

Description Deletes a file attachment from a document. Notes represents file attachments by file name and extension, not the full path.

Parameters

DOCHAND

The document containing the file attachment.

FILENAME

The filename of the attached file. Use htFileList to obtain a list of files within a document.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_DOCUMENT (invalid document handle);

HTFAIL_INVALID_FILE_ITEM (no such file attachment in the document).

Example

```
HTSTATUS status;
status = htFileDelete (dochand, "filename.txt");
```

See Also htFileList, htFilePut

htFileFetch

Summary Extracts a file attachment from a document to a file.

Syntax

```
HTSTATUS htFileFetch (dochand, directory, filename);
HTDOCHANDLE         dochand;       /* Input */
char           *directory;    /* Input, Optional */
char           *filename;          /* Input */
```

Description Extracts a file attachment from a document to an operating system file. A directory into which to extract the file may be specified.

Parameters

DOCHAND

The document containing the file attachment.

DIRECTORY

The directory into which to extract the file. A NULL pointer or empty string directs HiTest to use the current working directory. Valid directory values are a fully specified path with or without the drive (e.g., “C:\HITEST\DOC” or “\HITEST\DOC”), or a path relative to the current directory (e.g., “DOC” when the current directory is HITEST).

FILENAME

The filename of the attached file. Use htFileList to obtain a list of files within a document.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_DOCUMENT (invalid document handle);

HTFAIL_INVALID_FILE_ITEM (no such file attachment in the document).

HTFAIL_INVALID_DIRECTORY (extraction directory is invalid).

Example

```
HTSTATUS status;
status    =    htFileFetch    (dochand,    "C:\NOTES",
"filename.txt")
```

See Also htFileList

htFileList

Summary Iterates through file attachments in a document.

Syntax

```
HTSTATUS htFileList (dochand, first, filename);
HTDOCHANDLE         dochand;       /* Input */
HTBOOL          first;         /* Input */
char            *filename;          /* Output */
```

Description Returns the first or next file attachment filename from the list of file attachments in the document. Use this filename with the other htFile functions to manipulate file attachments.

Parameters

DOCHAND

The document from which to list file attachments.

FIRST

Whether to get the first or next file attachment. TRUE resets the attachment list, FALSE simply obtains the next attachment in the list. The value always acts as TRUE on the first call for a given document handle.

FILENAME

The buffer to receive the filename string. Attached files store only the file name and extension, not the full path. The constant HTLEN_FILENAME defines the maximum filename length.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_DOCUMENT (invalid document handle);

HTFAIL_END_OF_DATA (no more files).

Example

```
char filename [HTLEN_FILENAME + 1];
HTSTATUS status;
status = htFileList (dochand, FALSE, filename);
```

See Also htFileFetch

htFilePut

Summary Attaches a file to a document.

Syntax

```
HTSTATUS htFilePut (dochand, filepath);
HTDOCHANDLE          dochand;       /* Input */
char            *filepath;          /* Input */
```

Description Attaches a file to a document. Once attached, Notes represents file attachments by file name and extension, not the full path.

Parameters

DOCHAND

The document containing the file attachment.

FILENAME

The filename of the file to attach. If the file is not in the current working directory, use the full path.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_DOCUMENT (invalid document handle).

Example

```
HTSTATUS status;
status = htFilePut (dochand, "C:\WINDOWS\NOTES.INI");
```

See Also htFileFetch, htFileDelete

Font

Notes uses fonts in various objects, including composite records, views, and columns. A font does not support any functions, but does need documentation. Notes normally supports three basic fonts (roman, swiss, and typewriter). Any document which uses fonts beyond this set contains a composite item named "$FONTS" to define the additional fonts. HiTest provides operating system-specific font representation of fonts (currently available only for Windows) with the functions htCompGetOSFont and htCompPutOSFont. These functions simplify manipulation of font information within a document. A font contains four components: font face, font attributes, font color, and font size.

The face component describes the basic font. This component is either one of the values in the following list or a number equal to or greater than the constant HTFONT_FACE_USERDEF_MIN:

HTFONT_FACE_ROMAN Default
HTFONT_FACE_SWISS
HTFONT_FACE_TYPEWRITER

The following flags define font attributes in the HTFONT structure:
HTFONT_ATTRIB_BOLD
HTFONT_ATTRIB_ITALIC
HTFONT_ATTRIB_UNDERLINE
HTFONT_ATTRIB_STRIKEOUT
HTFONT_ATTRIB_SUPER
HTFONT_ATTRIB_SUB

The following constants define font color in the HTFONT structure:
HTFONT_COLOR_BLACK Default
HTFONT_COLOR_WRITE
HTFONT_COLOR_RED
HTFONT_COLOR_GREEN
HTFONT_COLOR_BLUE
HTFONT_COLOR_MAGENTA
HTFONT_COLOR_YELLOW
HTFONT_COLOR_CYAN

Each font has a size in points. Since this is simply an integer number, the only constant defined is the default constant HTFONT_SIZE_DEFAULT, which is ten.

The following structure represents a font:

```
typedef struct
{
   HTBYTE face;      /* Font face (HTFONT_FACE_xxx) */
   HTBYTE attrib;         /*  Font  attributes  (HTFONT_ATTRIB_xxx)
*/
   HTBYTE color;          /* Font color (HTFONT_COLOR_xxx) */
   HTBYTE size;      /* Font size in points */
} HTFONT;
```

Form

A form is one of two primary types of metadata (the other is a view). Each database contains zero or more forms, which describe the format of documents. A form consists of various attributes and one or more fields. HiTest can automatically filter data through forms, providing a more consistent representation of data. The filtering occurs only when the strict binding option is active (the default state), in which case HiTest type-checks all data transferred against the relevant form. Lotus Notes represents forms as simply data, and the standard Lotus Notes API supplies no abstraction of this data. The HiTest form abstraction accurately represents forms as metadata, and supports easy access to that metadata. The primary attributes of a form are a name and ID. HiTest uses the constant NULLID to represent an invalid form ID.

The following flags define form attributes in the HTFORM structure:

HTFORM_USE_REFERENCE	Use reference note
HTFORM_MAIL_ON_SAVE	Mail when saving document
HTFORM_RESPOSE_TO_RESP	Save REFID to response
HTFORM_RESPONSE_TO_DOC	Save REFID to main parent
HTFORM_RECALC_FIELDS	Recalc fields when focus is lost
HTFORM_FORM_IN_DOC	Store form in document
HTFORM_USE_FORE_COLOR	Use foreground color to paint
HTFORM_OLE_ACT_COMP	Activate OLE objects at compose
HTFORM_OLE_ACT_EDIT	Activate OLE objects at edit
HTFORM_OLE_ACT_READ	Activate OLE objects at read
HTFORM_SHOW_WIN_COMP	Show editor window if OLE_ACT_COMP
HTFORM_SHOW_WIN_EDIT	Show editor window if OLE_ACT_EDIT
HTFORM_SHOW_WIN_READ	Show editor window if OLE_ACT_READ
HTFORM_UPDATE_IS_RESP	Updates become responses
HTFORM_UPDATE_IS_PARENT	Updates become parents

htFormList returns the following form summary structure:

```
typedef struct
{
   HTFORMID formid;            /* Form ID */
   HTBOOL hidden;                  /* Whether form is hidden in
UI */
   char name [HTLEN_DESIGNNAME + 1];               /* Form name */
   char display_name1 [HTLEN_DISPLAYNAME + 1];     /*      Primary
display
                                                   name */
   char display_name2 [HTLEN_DISPLAYNAME + 1];     /*    Secondary
display
                                                   name */
} HTFORMSUMM;                 /* htFormList summary structure */
```

htFormGetAttrib returns the following form attribute structure:

```
typedef struct
{
   HTFLAGS flags;                   /* Form flags (HTFORM_xxx) */
```

```
   WORD color;                /* Background color */
   HTBOOL hidden;                  /* Whether form is hidden in
UI */
   char name [HTLEN_DESIGNNAME + 1];              /* Form name */
   char display_name1 [HTLEN_DISPLAYNAME + 1];    /*      Primary
display
                                                  name */
   char display_name2 [HTLEN_DISPLAYNAME + 1];    /*    Secondary
display
                                                  name */
} HTFORM;                                  /*     Form     attribute
structure */
```

The three name fields in the HTFORM and HTFORMSUMM structures handle Notes' multiple naming of objects. Forms may have multiple names, and the first name may consists of two parts. The *name* field contains the string which Notes uses internally to refer to a given form. The *display_name1* field contains the name which appears in the Notes UI. For a cascading form name, the *display_name2* field contains the cascading component of the Notes UI name. When a form has only one name, the *name* field is equal to either *display_name1* (if not cascading) or *display_name1*/*display_name2* (if cascading). The hidden field indicates whether the Notes UI Compose menu normally displays the form. A hidden form has either its display name enclosed in parenthesis or the form attribute "Include in Compose Menu" unchecked.

The form group contains the following functions:

htFormCopy	Copies a form from one cursor to another
htFormDelete	Deletes a form from a database
htFormGetAttrib	Obtains the attributes of a form
htFormGetId	Obtains a form ID from the form name
htFormList	Iterates through forms in a database
htFormSet	Assigns the active form for a cursor
htFormTemplate	Creates a virtual form from the next inserted document's bindings

htFormCopy

Summary Copies a form from one cursor to another.

Syntax

```
HTSTATUS    htFormCopy    (src_cursor,    src_formid,
dest_cursor,
                    dest_formname,  dest_formid);
HTCURSOR  src_cursor;          /* Input */
HTFORMID  src_formid;          /* Input */
HTCURSOR  dest_cursor;         /* Input */
char      *dest_formname;          /*  Input,  Optional
*/
FORMID    *dest_formid;        /* Output, Optional */
```

Description Copies a form between cursors, optionally assigning a new name. Notes requires form names within a database to be unique.

Parameters

SRC_CURSOR

The cursor from which to copy the form.

SRC_FORMID

The form to copy within the source cursor.

DEST_CURSOR

The cursor into which to copy the new form.

DEST_FORMNAME

The name for the new form in the destination cursor. To keep the original name, use NULL or the empty string. Otherwise, the name formatting follows the Notes UI rules (“display_name1\ display_name2 | name” -- see the Lotus Notes Application Developer’s Reference). A new name must be supplied when the source and destination cursors are connected to the same database.

DEST_FORMID

The buffer which receives the form ID for the new form.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_INVALID_FORM (source form does not exist);

HTFAIL_DUPLICATE (a form exists in the destination cursor with the same title);

HTFAIL_OVERFLOW (new form title is too long).

Example

```
HTFORMID new_formid;
HTSTATUS status;
status = htFormCopy (cursor1, formid, cursor2,
"NewForm",
                    &new_formid);
```

See Also htFormGetId, htFormDelete

htFormDelete

Summary Deletes a form from a database.

Syntax

```
HTSTATUS htFormDelete (cursor, formid);
HTCURSOR  cursor;         /* Input */
HTFORMID  formid;         /* Input */
```

Description Deletes a form from a database. A cursor's active form cannot be deleted.

Parameters

CURSOR

The cursor containing the form.

FORMID

The form to delete.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_INVALID_FORM (form does not exist);

HTFAIL_ACTIVE_RESULT (cannot delete the active form).

Example

```
HTSTATUS status;
status = htFormDelete (cursor, formid);
```

See Also htFormGetId, htFormCopy, htFormSet

htFormGetAttrib

Summary Obtains the attributes of a form.

Syntax

```
HTSTATUS htFormGetAttrib (cursor, formid, form);
HTCURSOR  cursor;         /* Input */
HTFORMID  formid;         /* Input */
HTFORM    *form;          /* Output */
```

Description Obtains complete attributes for a form.

Parameters

CURSOR

The cursor containing the form.

FORMID

The form for which to obtain attributes.

FORM

The structure to receive form attributes. See the Form object section preceding the htForm functions for a description of this structure and its contents.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_INVALID_FORM (form does not exist).

Example

```
HTFORM form;
HTSTATUS status;
status = htFormGetAttrib (cursor, formid, form);
```

See Also htFormList, htFormGetId

htFormGetId

Summary Obtains a form ID from the form name.

Syntax

```
HTFORMID htFormGetId (HTCURSOR cursor, char *formname);
HTCURSOR  cursor;        /* Input */
char      *formname;          /* Input */
```

Description Given a form name, obtains the form ID of the indicated form.

Parameters

CURSOR

The cursor containing the form.

FORMNAME

The form name for which to obtain the ID.

Returns HTFORMID for the requested form. Returns NULLID if the form does not exist.

Example

```
HTFORMID formid;
formid = htFormGetId (cursor, "Memo");
```

See Also htFormList, htFormGetAttrib

htFormList

Summary Iterates through forms in a database.

Syntax

```
HTSTATUS htFormList (cursor, operation, formsumm);
HTCURSOR  cursor;        /* Input */
HTLIST    operation;          /* Input */
HTFORMSUMM     *formsumm;          /* Output */
```

Description Returns the first or next form summary information from the list of forms in the cursor's database.

Parameters

CURSOR

The cursor from which to list forms.

OPERATION

An element of the HTLIST enumeration that indicates whether and how to reset the form list. Use HTLIST_REFRESH to discard the form list and obtain a new list from Notes. Use HTLIST_FIRST to set the next element in the list to the first element. Use HTLIST_NEXT to obtain the element following the previously fetched element. The first call to this function after opening the cursor always uses the value HTLIST_REFRESH.

HTFORMSUMM

The structure to receive the form's summary information. See the Form object section preceding the htForm functions for a description of this structure and its contents.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_END_OF_DATA (no more forms).

Example

```
HTFORMSUMM formsumm;
HTSTATUS status;
status = htFormList (cursor, HTLIST_NEXT, &formsumm);
```

See Also htFormGetAttrib

htFormSet

Summary Assigns the active form for a cursor.

Syntax

```
HTSTATUS htFormSet (HTCURSOR cursor, HTFORMID formid);
HTCURSOR  cursor;         /* Input */
HTFORMID  formid;         /* Input */
```

Description Sets the active form for a cursor, which is required when strict binding is in effect to filter results. When producing a flat index, the index only includes documents of this form. When loading or storing items, the items must exist in this form and be of the proper type. The active form is only used when strict binding is in effect. Use the htCurGetInfo function to obtain a cursor's active form. This function is invalid in a cursor containing an active index.

Parameters

CURSOR

The cursor in which to set the active form.

FORMID

The form to set as the active form. Use NULLID to clear the active form.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_INVALID_FORM (form does not exist);

HTFAIL_ACTIVE_RESULT (cannot set the active form with an active index)

Example

```
HTSTATUS status;
status = htFormSet (cursor, "Memo")
```

See Also htFormGetId, htCurSetOption, htCurGetInfo, htFormulaExec, htItemBind

htFormTemplate

Summary Creates a virtual form from the next inserted document's bindings.

Syntax

```
HTSTATUS htFormTemplate (cursor, formname);
HTCURSOR  cursor;         /* Input */
char      *formname;          /* Input */
```

Description Defines a virtual form which is created at the next htDocPut call. This function is invalid in a cursor containing an active index. After calling this function, any htItemBind calls create virtual items in the virtual form. The next htDocPut call creates a form from the virtual form containing the same items as the new document.

Parameters

CURSOR

The cursor in which to create the form template.

FORMNAME

The form name to use for the new form. The name formatting follows the Notes UI rules ("display_name1\display_name2 | name" -- see the Lotus Notes Application Developer's Reference).

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_DUPLICATE (a form with the given form name exists in this cursor);

HTFAIL_ACTIVE_RESULT (clear the active result to enable this operation).

Example

```
HTSTATUS status;
status = htFormTemplate (cursor, "NewForm");
```

See Also htFormSet, htItemBind, htDocPut

Formula

Each cursor contains a formula buffer to construct formulas in one or more pieces. On execution, HiTest assembles the entire formula internally. Formula execution produces an index. An empty formula produces the same results as "SELECT @ALL". When producing a view-based index, the formula buffer must be empty. The syntax of Notes selection formulas is defined in the Lotus Notes Application Developer's Reference.

The formula group contains the following functions:

htFormulaConcat	Concatenates a string to the formula buffer
htFormulaConcatf	Concatenates a printf-style formatted string to the formula buffer
htFormulaCopy	Copies a portion of the current formula buffer
htFormulaExec	Executes the formula buffer and produces an index
htFormulaLength	Returns the length of the current formula buffer
htFormulaReset	Clears the formula buffer

htFormulaConcat

Summary Concatenates a string to the formula buffer.

Syntax

```
HTSTATUS htFormulaConcat (cursor, string);
HTCURSOR  cursor;         /* Input */
char      *string;        /* Input */
```

Description Concatenates a string to the cursor's formula buffer. This supports construction of a formula a piece at a time, or in a single call. Use the htFormulaConcatf function for printf-style string formatting. Both the htCurReset and htFormulaReset functions clear the formula buffer.

Parameters

CURSOR

The cursor to use.

STRING

The string to concatenate to the formula buffer.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR.

Example

```
HTSTATUS status;
status = htFormulaConcat (cursor, "SELECT @All")
```

See Also htFormulaConcatf, htFormulaReset, htCurReset

htFormulaConcatf

Summary Concatenates a printf-style formatted string to the formula buffer.

Syntax

```
HTSTATUS htFormulaConcatf (cursor, format, ...);
HTCURSOR  cursor;         /* Input */
char      *format;        /* Input */
```

Description Concatenates a printf-style formatted string to the cursor's formula buffer. This supports construction of a formula a piece at a time, or in a single call. This function supports multiple arguments and the format string is any valid printf format string. Use the htFormulaConcat function for simple (non-formatted) formula concatenation. Both the htCurReset and htFormulaReset functions clear the formula buffer.

Parameters

CURSOR

The cursor to use.

FORMAT

The printf-style format string to concatenate to the formula buffer.

... (VARIABLE PARAMETERS)

Variable parameters for the printf-style formatting.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR.

Example

```
HTSTATUS status;
status = htFormulaConcatf (cursor, "SELECT Form = \"%s\"",
                           formname)
```

See Also htFormulaConcat, htFormulaReset, htCurReset

htFormulaCopy

Summary Copies a portion of the current formula buffer.

Syntax

```
HTINT htFormulaCopy (cursor, start, count, buffer);
HTCURSOR  cursor;        /* Input */
HTINT          start;         /* Input, Optional */
HTINT          count;         /* Input, Optional */
char      *buffer;       /* Output */
```

Description Copies some or all of the cursor's formula buffer to a supplied buffer.

Parameters

CURSOR

The cursor to use.

START

The character index in the formula buffer from which to start the copy (the first character is index zero). If start is greater than the formula length, the result is the empty string.

COUNT

The number of characters to copy. Use zero to copy until the end of the formula buffer. If there are not enough characters, the copy stops at the last character.

BUFFER

Buffer into which to copy the formula fragment.

Returns HTINT number of bytes copied.

Example

```
char formula_fragment [20];
length    =    htFormulaCopy    (cursor,    0,    19,
formula_fragment);
```

See Also htFormulaLength

htFormulaExec

Summary Executes the formula buffer and produces an index.

Syntax

```
HTSTATUS htFormulaExec (HTCURSOR cursor);
HTCURSOR  cursor;         /* Input */
```

Description Produces an index in the cursor by executing the contents of the formula buffer as a Notes formula. The empty formula produces the same effects as the formula "SELECT @ALL". See the Lotus Notes Application Developer's Reference for a description of valid formula syntax. After execution, the htIndex functions support manipulation of the index produced. The type of index depends on whether there is an active view.

If there is no active view, then this function produces a flat index. HiTest executes the formula against the cursor's database, producing a list of documents. Any active form must be set before calling this function (any attempts to do so with an active index will fail). If strict binding is in effect, then this function requires an active form, and the index will only contain documents of the active form. After execution, navigation through the resulting index is possible with the functions htIndexNavigate, htIndexSetPos, or htIndexSetTreePos.

If there is an active view, then that view produces the hierarchical index. The formula buffer must be empty. This function will lose efficiency if the VIEW_POSITION cursor-level option is active. Access to view data with the htCell functions requires a view-based index.

Parameters CURSOR

The cursor in which to execute the formula buffer.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_FORM_UNAVAIL (search formula with strict bind requires an active form);

HTFAIL_INVALID_FORMULA (formula contains incorrect syntax).

Example

```
HTSTATUS status;
status = htFormulaExec (cursor);
```

See Also htFormulaConcat, htFormulaConcatf, htFormulaReset, htFormSet, htViewSet, htIndexNavigate, htIndexSetPos, htIndexSetTreePos, htCurReset, htSetOption

htFormulaLength

Summary Returns the length of the current formula buffer.

Syntax

```
HTINT htFormulaLength (cursor);
HTCURSOR  cursor;         /* Input */
```

Description Returns the string length of the cursor's current formula buffer.

Parameters CURSOR

The cursor to use.

Returns HTINT formula buffer length. Returns zero if the formula buffer is empty.

Example

```
HTINT length;
length = htFormulaLength (cursor);
```

See Also htFormulaConcat, htFormulaConcatf, htFormulaCopy

htFormulaReset

Summary Clears the formula buffer.

Syntax

```
HTSTATUS htFormulaReset (HTCURSOR cursor);
HTCURSOR  cursor;         /* Input */
```

Description Clears the cursor's formula buffer.

Parameters CURSOR

The cursor to use.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR.

Example

```
htFormulaReset (cursor);
```

See Also htCurReset

Index

Each cursor may contain one active index. An index is a set of documents produced by executing a formula against a database, executing a full text search query against a database or index, or accessing a view. An index may be flat or hierarchical (view-based). There are two ways to move through an index: navigation relative to the current position, and assignment of an absolute position. A cursor in which an index has been produced contains an active index. This prevents certain operations (e.g., setting the active form or view) without first clearing the active index. Full text search can produce a new index or refine an existing index.

Setting an absolute position within a view-based index has two usages. The default usage (when the VIEW POSITION option is FALSE) is to position based on top-level entries within the view. This enables rapid movement through a view-based index, and is much quicker than normal navigation when moving over large distances. Set the VIEW POSITION option to TRUE to allow programs to locate any position within the view by ordinal number. This method causes a significant slowdown when moving through a view-based index, and should generally be avoided. Large databases will aggravate the slowdown. The preferred method for view-based positioning is the default of using the top-level entries.

The index group contains the following functions:

htIndexCount	Obtains the number of documents in an index
htIndexGetInfo	Obtains a piece of information about the active index
htIndexGetPos	Obtains the current position in the index
htIndexGetTreePos	Obtains the current hierarchical position in a view-based index
htIndexNavigate	Navigates through the documents in an index
htIndexRefresh	Refreshes a view-based index
htIndexSearch	Performs a full text search
htIndexSetPos	Assigns the current position in the index
htIndexSetTreePos	Assigns the current hierarchical position in a view-based index

htIndexCount

Summary Obtains the number of documents in an index.

Syntax

```
HTINT htIndexCount (cursor);
HTCURSOR  cursor;         /* Input */
```

Description Obtains the number of elements in the cursor's active index. Use this count as the maximum index position for the htIndexGetPos and htIndexSetPos functions. For a view-based index, the count depends on the VIEW_POSITION option. When it is inactive (the default), this function obtains the number of top-level entries in the index. When it is active, this function obtains the total number of entries in the index.

Parameters CURSOR

The cursor containing the relevant index.

Returns HTINT number of document in the cursor's active index. Returns zero if there is no active index.

Example

```
HTINT count;
count = htIndexCount (cursor);
```

See Also htFormulaExec, htIndexGetPos, htIndexSetPos, htSetOption

htIndexGetInfo

Summary Obtains a piece of information about the active index.

Syntax

```
HTSTATUS htIndex (cursor, item, buffer);
HTCURSOR  cursor;        /* Input */
HTINDEXINFO    item;          /* Input */
void      *buffer;       /* Output */
```

Description Fetches one of various index information items into a supplied buffer. Each item has a data type, and the buffer must be sufficiently large to hold the result.

Parameters

CURSOR

The cursor containing the relevant index.

ITEM

One value from an enumeration of index items. Each item corresponds to a type (and length, for variable length types). The following table lists legal items with their corresponding data types and, where relevant, lengths:

constant	type
HTINDEXINFO_ISSELECT	HTBOOL
HTINDEXINFO_ISVIEWBASED	HTBOOL
HTINDEXINFO_ISFTSEARCH	HTBOOL
HTINDEXINFO_VIEWDEPTH	HTINT
HTINDEXINFO_FTSCORE	HTINT

ISSELECT indicates whether the cursor contains an active index;

ISVIEWBASED indicates whether the current index is view-based.

ISFTSEARCH indicates whether the current index is a result of full text search.

VIEWDEPTH obtains the hierarchical depth of the current view-based index entry.

FTSCORE obtains the full text search relevance score of the current full text index entry.

BUFFER

The buffer to receive the requested information. This buffer should be of sufficient length to handle the result.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_ILLEGAL_ENUM (invalid item).

Example

```
HTBOOL isviewbased;
HTSTATUS status;
status = htIndexGetInfo (HTINDEXINFO_ISVIEWBASED,
                         &isviewbased);
```

See Also htFormulaExec, htIndexSearch, htCurReset

htIndexGetPos

Summary Obtains the current position in the index.

Syntax

```
HTINT htIndexGetPos (cursor);
HTCURSOR  cursor;         /* Input */
```

Description Obtains the current position in the cursor's index. The first element's position is one. Use the htIndexCount function to get the last element's position.

Parameters CURSOR

The cursor containing the relevant index.

Returns HTINT one-based position in the active index. Returns zero if there is no active index.

Example

```
HTINT position;
position = htIndexGetPos (cursor);
```

See Also htIndexCount, htIndexSetPos

htIndexGetTreePos

Summary Obtains the current hierarchical position in a view-based index.

Syntax

```
HTSTATUS htIndexGetTreePos (cursor, length, position);
HTCURSOR  cursor;        /* Input */
HTINT          length;        /* Input, Optional */
char      *position;          /* Output */
```

Description Obtains the current hierarchical position in the cursor's view-based index. This position is in the format of a Notes @DocNumber value (e.g., "13.2.4"). The position represents a 'pointer' to an exact view element. Use htIndexSetTreePos to move directly to this position. This function is only available on view-based indices when the VIEW_POSITION option is inactive.

Parameters

CURSOR

The cursor containing the relevant index.

LENGTH

The length of the buffer which will receive the position string. A value of zero indicates that the buffer is of sufficient length to contain the result.

POSITION

The buffer to receive the hierarchical position string.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_DATA_UNAVAIL (no active view-based index);

HTFAIL_OVERFLOW (position string is too long for supplied buffer).

Example

```
char position [81];
HTSTATUS status;
status = htIndexGetTreePos (cursor, 80, position);
```

See Also htIndexSetTreePos, htIndexGetPos

htIndexNavigate

Summary Navigates through the documents in an index.

Syntax

```
HTSTATUS htIndexNavigate (cursor, direction, docid, indent);

HTCURSOR cursor;        /* Input */

HTNAV          direction;          /* Input */

HTDOCID   *docid;        /* Output, Optional */

HTINT          *indent;       /* Output, Optional */
```

Description Navigates through the cursor's index using a navigation style (direction). More styles are available for view-based indices than for flat indices. To retrieve each of the entries in an index, use the HTNAV_NEXT direction (since position zero is before the first entry, this will iterate through all entries in the index). Every entry in a flat index is a document. The entries (rows) in a view-based index can be of three types: a document, a category, or a totals row. Navigation loses efficiency against a view-based index when the view position option is active.

Parameters

CURSOR

The cursor containing the relevant index.

DIRECTION

The style of navigation to use. This value is an element of the HTNAV enumeration, combined with zero or more HTNAV flags. The following table defines HTNAV values:

DIRECTION	MEANING
HTNAV_NEXT	Locate the next document
HTNAV_END	Go to the last document
HTNAV_PEER*	Go to the next peer
HTNAV_CHILD*	Go to the next child
HTNAV_PARENT*	Go to the next parent
HTNAV_MAIN*	Go to the next main document
HTNAV_CURRENT	Don't change the position

FLAGS	MEANING
HTNAV_PEEK	When done, restore the position prior to call

HTNAV_BACKWARD	Navigate backwards (invalid with CHILD)
HTNAV_NOCATEGORY*	Skip category entries
HTNAV_NOVIEWTOTALS*	Skip totals entries

**values marked with an asterisk are only valid for hierarchical indices*

DOCID

The buffer to receive the document ID of the new entry. Special constant values indicate a category or totals row. The document ID value for a category row is the constant HTINDEX_DOCID_CATEGORY and the document ID value for a totals row is the constant HTINDEX_DOCID_VIEWTOTAL. These special values are negative, and all valid document are positive.

INDENT

The buffer to receive the view cell indentation of the new entry. This value is always zero for flat indices. For document rows in a view-based index, the top level document is zero, the first response is one, etc. For category rows, this value is the depth of cascading category indentation levels. A totals row has an indent value of zero. When rendering a view visually, the indentation determines the number of three space prefixes to add before the view row. To determine the hierarchical depth of an index entry, use the htIndexGetInfo function with the HTINDEXINFO_VIEWDEPTH item.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_ILLEGAL_ENUM (invalid direction);

HTFAIL_INVALID_NAVTYPE (direction is invalid with the active index);

HTFAIL_END_OF_DATA (no more results available);

HTFAIL_DATA_UNAVAIL (no active index) OR;

HTFAIL_DATA_UNAVAIL (direction current with invalid view-based index position).

Example

```
HTINT indent;
HTDOCID docid;
HTSTATUS status;
status = htIndexNavigate (cursor,
                          HTNAV_NEXT + HTNAV_BACKWARD,
                          &docid, &indent);
```

See Also htFormulaExec, htIndexGetInfo, htIndexSetPos, htIndexRefresh, htDocFetch, htDocUpdate, htSetOption

htIndexRefresh

Summary Refreshes a view-based index.

Syntax

```
HTSTATUS htIndexRefresh (cursor);
HTCURSOR  cursor;         /* Input */
```

Description Performs a view refresh for the cursor's index. This functionality is only available for view-based indices. Changes such as deleting a document which affect a view require either this function or recomputation with htFormulaExec to integrate them into the index. After refreshing the index, the function attempts to restore the index position prior to this call.

Parameters CURSOR

The cursor containing the relevant index.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_DATA_UNAVAIL (no active view-based index).

Example

```
HTSTATUS status;
status = htIndexRefresh (cursor);
```

See Also htFormulaExec

htIndexSearch

Summary Performs a full text search.

Syntax

```
HTSTATUS htIndexSearch (cursor, query, flags, limit);
HTCURSOR  cursor;        /* Input */
char      *query;        /* Input */
HTFLAGS   flags;         /* Input */
HTINT          limit;          /* Input, Optional */
```

Description Performs a full text search against the documents in the active index (or the entire database if there is no active index). The results produced are either a flat or view-based index, depending on the index prior to the search. For view-based results, the search index does not contain a hierarchy (i.e., there is only one level and hierarchical navigation styles don't work), but still supports access to cell data. The index produced replaces any previous index, but canceling the full text search index with the HTSEARCH_CANCEL flag restores the original index.

Results of a full text search are available with the standard htIndex functions. Hierarchical navigation styles (e.g., CHILD, PARENT, etc.) do not work. Use the htIndexGetInfo function with the HTINDEXINFO_FTSCORE item to obtain scores produced from the full text search (see HTSEARCH_SCORE flag below).

Parameters

CURSOR

The cursor containing the relevant index.

QUERY

The full text query to execute. See the Lotus Notes Application Developers Reference for the syntax of a full text search query.

FLAGS

The options which affect the full text search. The HTSEARCH flags may be OR-ed together, although the HTSEARCH_CANCEL flag overrides all others. The following table describes the valid HTSEARCH flags:

FLAGS	MEANING
HTSEARCH_SCORE	Produce relevance scores (see htIndexGetInfo)
HTSEARCH_SORT_DATE	Sort results by date (descending)

HTSEARCH_SORT_ASCEND	Sort results in ascending order (default descending)
HTSEARCH_STEM_WORDS	Stem words in search
HTSEARCH_REFINE	Refine a previous search. HiTest requires an active full text search index. If not used, HiTest destroys any existing full text search index and performs this search against the original non-full text search index (if any)
HTSEARCH_CANCEL	Cancel existing full text search index, restoring the original non-full text search index (if any)

LIMIT

The maximum number of results to produce. The search will stop after finding this number of documents. Use zero for no limit, in which case the search returns all documents. The maximum limit value is 65535.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_OVERFLOW (limit too large);

HTFAIL_DATA_UNAVAIL (cancel or refine requires an active full text index);

HTFAIL_END_OF_DATA (no documents found or query is the empty string).

Example

```
HTSTATUS status;
status = htIndexSearch (cursor, "x and y",
                        HTSEARCH_SORT_DATE, 100);
```

See Also htFormulaExec, htIndexGetInfo, htIndexNavigate, htIndexGetPos, htIndexSetPos, htCurReset

htIndexSetPos

Summary Assigns the current position in the index.

Syntax

```
HTSTATUS htIndexSetPos (cursor, position);
HTCURSOR  cursor;        /* Input */
HTINT          position;          /* Input */
```

Description Assigns the current position in the cursor's index. The first element's position is one. Use the htIndexCount function to get the last element's position. This function loses efficiency on a view-based index when the VIEW_POSITION option is active.

Parameters

CURSOR

The cursor containing the relevant index.

POSITION

The position to set in the index. One is the first index entry.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_DATA_UNAVAIL (no active index);

HTFAIL_END_OF_DATA (position is invalid).

Example

```
HTSTATUS status;
status = htIndexSetPos (cursor, 30);
```

See Also htIndexCount, htIndexGetPos, htSetOption

htIndexSetTreePos

Summary Assigns the current hierarchical position in a view-based index.

Syntax

```
HTSTATUS htIndexSetPos (cursor, position);
HTCURSOR  cursor;        /* Input */
char           *position;          /* Input */
```

Description Assigns the current hierarchical position in the cursor's index. The position is in the format of a Notes @DocNumber (e.g., "13.2.4"). Programs can either use htIndexGetTreePos to obtain the current position or construct a position string programatically. This function is only available on view-based indices when the VIEW_POSITION option is inactive.

Parameters

CURSOR

The cursor containing the relevant index.

POSITION

The hierarchical position to set in the index.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_DATA_UNAVAIL (no active view-based index);

HTFAIL_END_OF_DATA (position is invalid);

HTFAIL_OVERFLOW (multi-level position is invalid in a full-text search index).

Example

```
HTSTATUS status;
status = htIndexSetPos (cursor, "13.2.4");
```

See Also htIndexGetTreePos, htIndexSetPos, htIndexSearch

Item

Notes stores individual data values in items, which in turn make up documents. Notes uses both the terms ‘item’ and ‘field’ to describe data within a document. HiTest uses a single term for clarity, and item represents document data as opposed to form fields. The primary attributes of an item are an item name, a data type, and a data value. With HiTest, items can be accessed either from open documents, or by binding items to the results in an index. While Lotus Notes itself imposes no restrictions on items within a document, HiTest adds a minimal set of qualifications which support basic access and facilitate simpler item access.

HiTest allows only one item of a given name within a given document. This restriction does not apply to file attachments (which all use the same name) or composite items (which are stored in multiple items of the same name). Use the htFile functions to manipulate file attachments. HiTest represents all composite items of the same name as a single item. Use the htComp and htComprec functions to manipulate composite items.

Another optional restriction imposes a form of type-checking. When the STRICT_BIND option is in effect, all items in a document must match (name and data type) a field within that document’s form. Disable this option with either the htSetOption or htCurSetOption functions, STRICT_BIND option.

Several item functions use two data type parameters (type and itemtype) to implement automatic conversion. In these functions the type parameter is the data type of data as seen by the calling program. The itemtype parameter is the data type of the item within the document. HiTest performs any conversions automatically when loading data to or storing data from the calling program.

When storing an item into a document, HiTest obtains the item flags from the form field (if strict binding is active) and automatically stores them with the item. For example, using htItemPut to create a “SendTo” item in a document created with the “Memo” form would automatically get the item flag NAMES from the form field. HiTest sets this flag in the “SendTo” document item, indicating that the item contains distinguished names.

One item flag assigned independently from the form’s fields is the summary flag. For an item to be accessible from within a view, the item’s summary flag must be set. If the summary items for a single document exceed the value HTLEN_SUMMARY_DATA (15K), then the document may not display properly in views, and cell values may not be accessible. If the length of any single item exceeds HTLEN_COMPUTE_DATA (also 15K), then that item is not usable in a view or any other form of computation. By default, items of length less than HTDEFAULT_SUMMARY_LIMIT (8K) are stored with their summary flag set. Use the htSetOption or htCurSetOption functions to change this limit to any value up to HTLEN_SUMMARY_DATA.

The following flags define item attributes:

HTITEM_SIGN	Signed field
HTITEM_SEAL	Sealed field (encrypted)
HTITEM_SUMMARY	Summary field (usable in formulas)
HTITEM_READWRITERS	Author Names field
HTITEM_NAMES	Names field
HTITEM_PROTECTED	To edit field requires Editor access
HTITEM_READERS	Reader Names field

Use the HTNAME_REF constant (defined as “$REF”) with the HTTYPE_REF data type to retrieve and insert reference items for response documents.

htItemList returns the following form attribute structure:

```
typedef struct
{
   HTTYPE type;                 /* Item data type */
   char name [HTLEN_FIELDNAME + 1]; /* Item name */
   HTFLAGS flags;                    /* Item flags (HTITEM_xxx) */
   HTINT length;                     /* Item value length */
   void *value;                 /* Item value (if requested) */
} HTITEM;                       /* Item summary structure */
```

The item group contains the following functions:

htItemBind	Binds an item name to a program variable
htItemCount	Obtains the number of items in a document
htItemDelete	Deletes an item from a document
htItemFetch	Converts and retrieves the data for an item into a supplied buffer
htItemGetInfo	Obtains a piece of information about a document item
htItemGetPtr	Returns a pointer to an item's data converted to a specified type
htItemLength	Obtains the length of an item as converted to a specified data type
htItemList	Iterates through items in a document
htItemPut	Writes an item to a document, overwriting any existing item of the same name
htItemUnbind	Removes the binding of an item name

htItemBind

Summary Binds an item name to a program variable.

Syntax

```
HTSTATUS htItemBind (cursor, itemname, type, length,
                     itemtype, buffer, datalen, nullind,
                     update);
HTCURSOR  cursor;        /* Input */
char      *itemname;          /* Input */
HTTYPE    type;          /* Input, Optional */
HTINT          length;        /* Input, Optional */
HTTYPE    itemtype;           /* Input, Optional */
void      *buffer;       /* Input */
HTINT          *datalen;           /*  Input,  Optional
*/
HTBOOL    *nullind;           /* Input, Optional */
HTBOOL    *update;       /* Input, Optional */
```

Description Creates a 'binding' between a variable in the calling program and an itemname in the active index. This binding is used by the htDocFetch, htDocPut, and htDocUpdate functions. Use item binding when operating on the same set of items in multiple documents. Create bindings after using htFormulaExec to produce an index. Remove bindings with htFormulaExec, htCurReset, or htItemUnbind with the same item name.

Fetching a document causes all bound items to be converted and transferred from the document to the bound buffers. Updating a document writes certain bound items (those with their update parameter changed to TRUE) back to the last fetched document. Putting a document converts and transfers data from the bound buffers into items in the new document. Binding a composite item generates an error when the cursor option FETCH_SUMMARY is active, since only summary data is available.

Parameters

CURSOR

The cursor containing the desired index.

ITEMNAME

The name of the item to bind. When strict binding is in effect, this item must exist as a field in the active form.

TYPE

The data type for data in the supplied buffer. When fetching, HiTest converts the item data to this type before copying it into the buffer. When storing (with htDocPut or htDocUpdate), HiTest converts data from this type to the item's type before writing it to the document. This enables automatic conversion between an item's data and the supplied buffer. A value of zero directs HiTest to use the item's data type (if strict binding is inactive, the item's data type must be supplied in the itemtype parameter).

LENGTH

The maximum length of the supplied buffer. Use zero when a buffer is known to be of sufficient length. HiTest ignores this value when storing data (with htDocPut or htDocUpdate).

ITEMTYPE

The data type of the item. When strict binding is active, use either zero or the data type of the corresponding field in the form. A value of zero directs HiTest to use the item's data type (if strict binding is inactive, the item's data type must be supplied in the type parameter).

BUFFER

The buffer into which to fetch or store data for this item.

DATALEN

The buffer to use to specify a specific item's data length. When fetching the item, HiTest sets this value to the length of the data written to the data buffer. When storing the item (with htDocPut or htDocUpdate), set this value to nonzero to supply the length of the item value. A value of zero directs HiTest to determine the length. Use NULL to omit this functionality.

NULLIND

The buffer to use to specify a specific item's null indicator. When fetching an item not in the document, HiTest sets this value to TRUE (otherwise FALSE). When storing the item (with htDocPut or htDocUpdate), setting this value to TRUE results in a NULL item (i.e., HiTest does not store the item in the document) regardless of other values. Use NULL to omit this functionality.

UPDATE

The buffer to use to specify a specific item's update indicator. When fetching the item, HiTest sets this value to FALSE. Setting this value to TRUE and then calling htDocUpdate directs HiTest to write the new item value back to the fetched document. Bind multiple items with the same update indicator buffer to perform multiple item updates by setting a single update indicator to TRUE and calling htDocUpdate. Use NULL to omit this functionality.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR

HTFAIL_FORM_UNAVAIL (strict binding requires an active form);

HTFAIL_INVALID_FIELD (no such field exists in the active form);

HTFAIL_DATA_UNAVAIL (cannot bind a composite item if fetch_summary is active).

Example

```
char date_string [HTLEN_DATETIME_TEXT + 1];
HTBOOL nullind, update;
HTSTATUS status;
status = htItemBind (cursor, "Date", HTTYPE_TEXT, 0,
                     HTTYPE_DATETIME, date_string, NULL,
                     &nullind, &update);
```

See Also htItemUnbind, htDocFetch, htDocPut, htDocUpdate, htCellBind, htFormulaExec, htCurReset

htItemCount

Summary Obtains the number of items in a document.

Syntax

```
HTSTATUS htItemCount (dochand, itemcount);
HTDOCHANDLE          dochand;       /* Input */
HTINT                *itemcount;    /* Output */
```

Description Obtains the number of items in the document, including composite items and excluding file attachments. HiTest counts (and handles with the htComp functions) all composite items of the same name as one item. This count does not include file attachments, which are handled with the htFile functions.

Parameters

DOCHAND

The document from which to obtain the item count.

ITEMCOUNT

The buffer to receive the number of items in the document.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_DOCUMENT (invalid document handle).

Example

```
HTINT count;
HTSTATUS status;
status = htItemCount (dochand, &count);
```

See Also htItemList

htItemDelete

Summary Deletes an item from a document.

Syntax

```
HTSTATUS htItemDelete (dochand, itemname);
HTDOCHANDLE          dochand;        /* Input */
char            *itemname;           /* Input */
```

Description Deletes an item from a document. This function does not work on file attachments (use the htFile functions to manipulate or delete file attachments). Deleting a composite item deletes all items of that name from the document (HiTest considers all composite items of the same name as single item). Deleting an open composite item (i.e., represented by a valid composite handle) invalidates that composite handle.

Parameters

DOCHAND

The document containing the item.

ITEMNAME

The name of the item to delete. When strict binding is in effect, this item must exist as a field in the document's form.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_DOCUMENT (invalid document handle);

HTFAIL_INVALID_FIELD (field is not in the document's form);

HTFAIL_INVALID_ITEM (item does not exist in the document).

Example

```
HTSTATUS status;
status = htItemDelete (dochand, "Date");
```

See Also htDocClose

htItemFetch

Summary Converts and retrieves the data for an item into a supplied buffer.

Syntax

```
HTSTATUS htItemFetch (dochand, itemname, type, length,
                      itemtype, buffer);
HTDOCHANDLE          dochand;  /* Input */
char            *itemname;     /* Input */
HTTYPE          *type;    /* Input/Output, Optional */
HTINT                *length;  /* Input/Output, Optional
*/
HTTYPE          *itemtype;     /* Input/Output, Optional
*/
void            *buffer;  /* Output */
```

Description Transfers the item's data from a document item to a supplied buffer. If requested, HiTest converts the data before writing it to the buffer. Use htItemLength to determine the required buffer length. To have HiTest manage the buffer, use the similar htItemGetPtr function.

Parameters

DOCHAND

The document containing the item.

ITEMNAME

The name of the item to fetch. When strict binding is in effect, this item must exist as a field in the document's form.

TYPE

The data type representing the type of data retrieved -- HiTest converts the item to this type. A value of zero directs HiTest to use the item's data type (if strict binding is inactive, the item's data type must be supplied in the itemtype parameter) and to return the item's type in this location.

LENGTH

The length of the supplied buffer. A value zero or a NULL pointer indicates that the buffer is large enough to hold the result data. A value of zero directs HiTest to return the length of data retrieved in this location. Use htItemLength to determine the length before retrieving the data. A length which is insufficient to contain the result is valid only when the destination type is

HTTYPE_TEXT and the global option TEXT_TRUNCATE is active. In this case, the resulting text is truncated to fit in the buffer.

ITEMTYPE

The data type of the item. When strict binding is active, use either zero or the data type of the corresponding field in the form. When strict binding is inactive, use either zero or the data type of the item within the document. A value of zero directs HiTest to use the item's data type (if strict binding is inactive, the item's data type must be supplied in the type parameter) and to return the item's type in this location.

BUFFER

The buffer to receive the converted data value.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_DOCUMENT (invalid document handle);

HTFAIL_INVALID_FIELD (field is not in the document's form);

HTFAIL_INVALID_ITEM (item does not exist in the document);

HTFAIL_INVALID_CONVERT (item type does not convert to requested type);

HTFAIL_OVERFLOW (retrieved data does not fit in supplied buffer).

Example

```
char buffer [HTLEN_DATETIME_TEXT + 1];
HTTYPE type = HTTYPE_TEXT, itemtype = HTTYPE_DATETIME;
HTINT length = 0;
HTSTATUS status;
status = htItemFetch (dochand, "Date", &type, &length,
                      &itemtype, buffer);
```

See Also htItemLength, htItemGetPtr

htItemGetInfo

Summary Obtains a piece of information about a document item.

Syntax

```
HTSTATUS htItemGetInfo (dochand, itemname, item, buffer);

HTDOCHANDLE         dochand;       /* Input */

char           *itemname;          /* Input */

HTITEMINFO          item;          /* Input */

void           *buffer;       /* Output */
```

Description Fetches one of various item-level information items into a supplied buffer. Each item has a data type, and the buffer must be sufficiently large to hold the result.

Parameters

DOCHAND

The document containing the item.

ITEMNAME

The name of the desired item within the form.

ITEM

One value from an enumeration of item information values. Each value corresponds to a type (and length, for variable length types). The following table lists legal values with their corresponding data types and, where relevant, lengths:

constant	type
HTITEMINFO_TYPE	HTTYPE
HTITEMINFO_FLAGS	HTFLAGS
HTITEMINFO_LENGTH	HTINT

TYPE obtains the item's data type;

FLAGS obtains the item's flags;

LENGTH obtains the item's data length, in its stored format.

BUFFER

The buffer to receive the requested information. This buffer should be of sufficient length to handle the result.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_DOCUMENT (invalid document handle);

HTFAIL_INVALID_ITEM (item does not exist in the document);

HTFAIL_ILLEGAL_ENUM (invalid item parameter).

Example

```
HTFLAGS flags;

HTSTATUS status;

status = htItemGetInfo (dochand, "Date",
HTITEMINFO_FLAGS,

                    &flags);
```

See Also htItemList

htItemGetPtr

Summary Returns a pointer to an item's data converted to a specified type.

Syntax

```
HTSTATUS htItemGetPtr (dochand, itemname, type, length,
                       itemtype, buffer);
HTDOCHANDLE         dochand;  /* Input */
char           *itemname;     /* Input */
HTTYPE         *type;    /* Input/Output, Optional */
HTINT               *length;  /* Output, Optional */
HTTYPE         *itemtype;     /* Input/Output, Optional
*/
void           **buffer;      /* Output */
```

Description Transfers the item's data from a document item to a buffer allocated and managed by HiTest. If requested, HiTest converts the data before writing it to the buffer. The calling program cannot modify the contents of this buffer, and HiTest frees the buffer when closing the document. To fetch data into a buffer managed by the calling program, use the similar htItemFetch function.

Parameters

DOCHAND

The document containing the item.

ITEMNAME

The name of the item to fetch. When strict binding is in effect, this item must exist as a field in the document's form.

TYPE

The data type representing the type of data retrieved -- HiTest converts the item to this type. A value of zero directs HiTest to use the item's data type (if strict binding is inactive, the item's data type must be supplied in the itemtype parameter) and to return the item's type in this location.

LENGTH

The buffer to receive the length of the retrieved data.

ITEMTYPE

The data type of the item. When strict binding is active, use either zero or the data type of the corresponding field in the form. When strict binding is inactive, use either zero or the data type of the item within the document. A value of zero directs HiTest to use the item's data type (if strict binding is inactive, the item's data type must be supplied in the type parameter) and to return the item's type in this location.

BUFFER

The buffer to receive the pointer to the converted data value. Do not modify or free this data.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_DOCUMENT (invalid document handle);

HTFAIL_INVALID_FIELD (field is not in the document's form);

HTFAIL_INVALID_ITEM (item does not exist in the document);

HTFAIL_INVALID_CONVERT (item type does not convert to requested type).

Example

```
char *buffer;
HTTYPE type = HTTYPE_TEXT, itemtype = HTTYPE_DATETIME;
HTINT length;
HTSTATUS status;
status = htItemGetPtr (dochand, "Date", &type, &length,
                       &itemtype, &buffer);
```

See Also htItemFetch

htItemLength

Summary Obtains the length of an item as converted to a specified data type.

Syntax

```
HTSTATUS htItemLength (dochand, itemname, type, itemtype,
                       length);
HTDOCHANDLE         dochand;  /* Input */
char           *itemname;     /* Input */
HTTYPE         *type;    /* Input/Output, Optional */
HTTYPE         *itemtype;     /* Input/Output, Optional
*/
HTINT               *length;  /* Output */
```

Description Obtains the length of an item's data as converted to a specified data type. Use this length to provide a buffer of the proper length for htItemFetch.

Parameters

DOCHAND

The document containing the item.

ITEMNAME

The name of the item whose length is to be determined. When strict binding is in effect, this item must exist as a field in the document's form.

TYPE

The data type representing the destination type -- the length returned is the length of the item as converted to this type. A value of zero directs HiTest to use the item's data type (if strict binding is inactive, the item's data type must be supplied in the itemtype parameter) and to return the item's type in this location.

ITEMTYPE

The data type of the item. When strict binding is active, use either zero or the data type of the corresponding field in the form. When strict binding is inactive, use either zero or the data type of the item within the document. A value of zero directs HiTest to use the item's data type (if strict binding is inactive, the item's data type must be supplied in the type parameter) and to return the item's type in this location.

LENGTH

The buffer to receive the data length. This is the length of the data as converted to the requested type.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_DOCUMENT (invalid document handle);

HTFAIL_INVALID_FIELD (field is not in the document's form);

HTFAIL_INVALID_ITEM (item does not exist in the document).

Example

```
HTTYPE type = HTTYPE_TEXT, itemtype = HTTYPE_DATETIME;
HTINT length;
HTSTATUS status;
status = htItemLength (dochand, "Date", &type, &itemtype,
                       &length);
```

See Also htItemFetch

htItemList

Summary Iterates through items in a document.

Syntax

```
HTSTATUS htItemList (dochand, first, getvalue, item);
HTDOCHANDLE         dochand;       /* Input */
HTBOOL          first;         /* Input */
HTBOOL          getvalue;           /* Input */
HTITEM          *item;         /* Output */
```

Description Returns the first or next item information from the list of items in the document. This function also optionally obtains a read-only pointer to the item's value.

Parameters

DOCHAND

The document from which to list items.

FIRST

Whether to get the first or next item. TRUE resets the item list, FALSE simply obtains the next item in the list. The value is always TRUE on the first call for a given document handle.

GETVALUE

Whether to obtain a read-only pointer to the item's value. The pointer is in the HTITEM structure, and must not be modified or freed by the calling program. The pointer becomes invalid when the document is closed.

ITEM

The structure to receive information on the item. The item information consists of the item name, item data type, item flags, and length and value pointer. Use a getvalue parameter value of TRUE to retrieve the length and value results (otherwise they are zero and NULL, respectively).

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_DOCUMENT (invalid document handle);

HTFAIL_END_OF_DATA (no more items).

Example

```
HTITEM item;
```

```
HTSTATUS status;
status = htItemList (dochand, FALSE, TRUE, &item);
```

See Also htItemCount, htItemGetInfo

htItemPut

Summary Writes an item to a document, overwriting any existing item of the same name.

Syntax

```
HTSTATUS htItemPut (dochand, itemname, type, length,
                    itemtype, buffer);
HTDOCHANDLE          dochand;       /* Input */
char            *itemname;          /* Input */
HTTYPE          type;          /* Input */
HTINT                length;        /*  Input,  Optional
*/
HTTYPE          itemtype;           /*  Input,  Optional
*/
void            *buffer;       /* Input */
```

Description Writes data from a buffer to a document item. If requested, HiTest converts the data before writing it to the document. This function deletes any existing item of the same name.

Parameters

DOCHAND

The document to receive the new item.

ITEMNAME

The name of the item to put. When strict binding is in effect, this item must exist as a field in the document's form.

TYPE

The data type representing the type of data in the buffer -- HiTest converts the data from this type to the item's type.

LENGTH

The length of the supplied data. A value zero directs HiTest to determine the length.

ITEMTYPE

The data type of the item. When strict binding is active, use either zero or the data type of the corresponding field in the form. When strict binding is inactive, use either zero or the data type of

the item within the document. A value of zero directs HiTest to use the item's data type (if strict binding is inactive, the item's data type is supplied in the type parameter).

BUFFER

The buffer containing the item's new value.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_DOCUMENT (invalid document handle);

HTFAIL_INVALID_FIELD (field is not in the document's form);

HTFAIL_INVALID_CONVERT (type does not convert to item type).

Example

```
HTSTATUS status;
status = htItemPut (dochand, "Date", HTTYPE_TEXT, 0,
                    HTTYPE_DATETIME, "1/29/66");
```

See Also htItemFetch, htDocClose

htItemUnbind

Summary Removes the binding of an item name.

Syntax

```
HTSTATUS htItemUnbind (cursor, itemname);
HTCURSOR  cursor;         /* Input */
char      *itemname;          /* Input */
```

Description Cancels the effects of any htItemBind performed with the same item name. Producing a new index automatically cancels all bindings.

Parameters

CURSOR

The cursor used in the binding operation.

ITEMNAME

The name of the item used in the binding operation.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR

HTFAIL_INVALID_ITEM (no such item bound).

Example

```
HTSTATUS status;
status = htItemUnbind (cursor, "Date");
```

See Also htItemBind, htDocFetch, htCurReset

Macro

A macro is a stored set of formulas which perform an action. Each database contains zero or more macros. A macro runs against either all or some of the documents in a database. In addition, a macro may select or modify existing documents or create new documents. A search macro also performs a full text search. The primary attributes of a macro are a name and ID. HiTest uses the constant NULLID to represent an invalid macro ID.

htMacroList returns the following macro summary structure:

```
typedef struct
{
   HTMACROID macroid;                /* Macro ID */
   HTBOOL hidden;                    /* Whether macro is hidden in
UI */
   char name [HTLEN_DESIGNNAME + 1];              /*  Macro  name
*/
   char display_name1 [HTLEN_DISPLAYNAME + 1];     /*      Primary
display
                                                  name */
   char display_name2 [HTLEN_DISPLAYNAME + 1];     /*    Secondary
display
                                                  name */
} HTMACROSUMM;                  /* htMacroList summary structure */
```

The three name fields in the HTMACROSUMM structure handles Notes' multiple naming of objects. Macros may have multiple names, and the first name may consists of two parts. The *name* field contains the string which Notes uses internally to refer to a given macro. The *display_name1* field contains the name which appears in the Notes UI. For a cascading macro name, the *display_name2* field contains the cascading component of the Notes UI name. When a macro has only one name, the *name* field is equal to either *display_name1* (if not cascading) or *display_name1*/*display_name2* (if cascading). The hidden field indicates whether Notes normally displays the macro in the Notes UI Run-Macro menu. A hidden macro has its "Include in 'Tools Run Macros' Menu" option unchecked.

The macro group contains the following functions:

htMacroCopy	Copies a macro from one cursor to another
htMacroDelete	Deletes a macro from a database
htMacroExec	Executes a macro
htMacroGetId	Obtains a macro ID from the macro name
htMacroList	Iterates through macros in a database

htMacroCopy

Summary Copies a macro from one cursor to another.

Syntax

```
HTSTATUS    htMacroCopy    (src_cursor,    src_macroid,
dest_cursor,
                    dest_macroname, dest_macroid);
HTCURSOR  src_cursor;          /* Input */
HTMACROID src_macroid;         /* Input */
HTCURSOR  dest_cursor;         /* Input */
char      *dest_macroname;          /*  Input,  Optional
*/
MACROID   *dest_macroid;            /* Output, Optional
*/
```

Description Copies a macro between cursors, optionally assigning a new name. Notes requires macro names within a database to be unique.

Parameters

SRC_CURSOR

The cursor from which to copy the macro.

SRC_MACROID

The macro to copy within the source cursor.

DEST_CURSOR

The cursor into which to copy the new macro.

DEST_MACRONAME

The name for the new macro in the destination cursor. To keep the original name, use NULL or the empty string. Otherwise, the name formatting follows the Notes UI rules (“display_name1\ display_name2 | name” -- see the Lotus Notes Application Developer’s Reference). A new name is required when the source and destination cursors are the same.

DEST_MACROID

The buffer which receives the macro ID for the new macro.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_INVALID_MACRO (source macro does not exist);

HTFAIL_DUPLICATE (a macro exists in the destination cursor with the same title);

HTFAIL_OVERFLOW (new macro title is too long).

Example

```
HTMACROID new_macroid;
HTSTATUS status;
status = htMacroCopy (cursor1, macroid, cursor2,
"NewMacro",
                      &new_macroid);
```

See Also htMacroGetId, htMacroDelete

htMacroDelete

Summary Deletes a macro from a database.

Syntax

```
HTSTATUS htMacroDelete (cursor, macroid);
HTCURSOR  cursor;         /* Input */
HTMACROID macroid;        /* Input */
```

Description Deletes a macro from a database.

Parameters

CURSOR

The cursor containing the macro.

MACROID

The macro to delete.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_INVALID_MACRO (macro does not exist);

Example

```
HTSTATUS status;
status = htMacroDelete (cursor, macroid);
```

See Also htMacroGetId, htMacroCopy

htMacroExec

Summary Executes a macro.

Syntax

```
HTSTATUS htMacroExec (cursor, macroid, total_count,
                      action_count);
HTCURSOR  cursor;        /* Input */
HTMACROID macroid;       /* Input */
HTINT          *total_count;  /* Output, Optional */
HTINT          *action_count; /* Output, Optional */
```

Description Executes a macro within a cursor. Notes macros contain execution parameters and a formula. The execution parameters define where to run the macro, the documents on which to run the macro, and how to affect documents. Where to run a macro (e.g., from the Notes UI menu; as a background macro; etc.) is not relevant to this function. The other options have a significant effect on this function.

All macros define a set of documents on which to run. The following list defines the effect of this option:

11. Run on all documents in database: Run the macro on all documents in the database. This option operates the same in HiTest as in the Notes UI.

12. Run on documents not yet processed by macro: Run the macro on all documents in the database on which it has not previously been run. This option operates the same in HiTest as in the Notes UI.

13. Run on documents not yet marked read by you: Run the macro on all documents in the database, since Notes user unread information is only available in the Notes UI. This option does not operate the same in HiTest as in the Notes UI.

14. Run on all documents in view: Run the macro on all documents in the current view-based index. Requires an active view-based index. This option operates the same in HiTest as in the Notes UI. Search macros created from the Notes UI use this option.

15. Run on selected documents in view: Run the macro on all documents in the current index, since selected information is a property of the Notes UI. Run against either a view-based or flat index, or against the entire database if there is no active index. This option does not operates the same in HiTest as in the Notes UI.

All macros define an operation to perform on a subset of documents which the macro runs against. The following list describes the effect of this option:

16. Update existing document when run: Modify documents affected by the macro. This option operates the same in HiTest as in the Notes UI.

17. Select document when run: Produce a new flat index from documents selected by the macro. This option destroys any existing index. This option provides the same basic selection functionality as in the Notes UI, differing only in the presentation of results (on-screen selection is a property of the Notes UI).

18. Create new document when run: Create new documents with relevant modifications from documents affected by the macro. This option operates the same in HiTest as in the Notes UI.

Due to anomalous behavior in the standard Notes V3 API, macros which perform document deletion (i.e., use the @DeleteDocument function) may not delete documents. To work around this problem, rewrite the macro formula and embed the deletion in an @If function. For example, the formula

```
SELECT Form = "FormName"; @DeleteDocument"
```

will not work, but the formula

```
SELECT @If (Form = "FormName"; @DeleteDocument; "")
```

will successfully delete the proper documents.

Parameters

CURSOR

The cursor containing the macro.

MACROID

The macro to execute.

TOTAL_COUNT

Buffer to receive the total number of documents selected by the macro. For macros which modify documents, this is not necessarily equivalent to the total number of documents affected (see action_count parameter).

ACTION_COUNT

Buffer to receive the total number of documents acted on by the macro. For a selection macro, this is the same as total_count. For macros which modify or create documents, this is the number of documents modified or created, which is equal to or less than the value for the total_count parameter.

Returns

HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_INVALID_MACRO (macro does not exist or is invalid);

HTFAIL_DATA_UNAVAIL (macro requires an active index);

Example

```
HTINT total_count, action_count;
HTSTATUS status;
status = htMacroExec (cursor, macroid, &total_count,
                      &action_count);
```

See Also htMacroList, htMacroGetId, htViewSet, htFormulaExec, htIndexNavigate

htMacroGetId

Summary Obtains a macro ID from the macro name.

Syntax

```
HTMACROID htMacroGetId (HTCURSOR cursor, char *macroname);
HTCURSOR  cursor;        /* Input */
char      *macroname;    /* Input */
```

Description Given a macro name, obtains the macro ID of the indicated macro.

Parameters

CURSOR

The cursor containing the macro.

MACRONAME

The macro name for which to obtain the ID.

Returns HTMACROID for the requested macro. Returns NULLID if the macro does not exist.

Example

```
HTMACROID macroid;
macroid = htMacroGetId (cursor, "Memo");
```

See Also htMacroList, htMacroExec

htMacroList

Summary Iterates through macros in a database.

Syntax

```
HTSTATUS htMacroList (cursor, operation, macrosumm);
HTCURSOR       cursor;        /* Input */
HTLIST         operation;          /* Input */
HTMACROSUMM          *macrosumm;    /* Output */
```

Description Returns the first or next macro summary information from the list of macros in the cursor's database.

Parameters

CURSOR

The cursor from which to list macros.

OPERATION

An element of the HTLIST enumeration that indicates whether and how to reset the macro list. Use HTLIST_REFRESH to discard the macro list and obtain a new list from Notes. Use HTLIST_FIRST to set the next element in the list to the first element. Use HTLIST_NEXT to obtain the element following the previously fetched element. The first call to this function after opening the cursor always uses the value HTLIST_REFRESH.

HTMACROSUMM

The structure to receive the macro's summary information. See the Macro object section preceding the htMacro functions for a description of this structure and its contents.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_END_OF_DATA (no more macros).

Example

```
HTMACROSUMM macrosumm;
HTSTATUS status;
status = htMacroList (cursor, HTLIST_NEXT, &macrosumm);
```

See Also htMacroGetId, htMacroExec

Mail

Mail within Notes is usually the act of sending *documents* of the form "memo". Notes can actually send any *document*. The mail functions simplify sending mail *documents*.

The following flags affect mail operations:

HTMAIL_PRIORITY_LOW	Set mail priority to low
HTMAIL_PRIORITY_HIGH	Set mail priority to high
HTMAIL_REPORT_NONE	No delivery report
HTMAIL_REPORT_CONFIRM	Confirmed delivery report
HTMAIL_RETURN_RECEIPT	Request return receipt
HTMAIL_SAVE	Save mail document
HTMAIL_SAVE_MAILDB	Save mail document in Notes user's mail database
HTMAIL_BOUND_ITEMS	Add bound items to mail document
HTMAIL_EMBED_FORM	Embed form within mail document

The following item name constants define common mail items:

HTMAIL_ITEM_SENDTO	SendTo
HTMAIL_ITEM_COPYTO	CopyTo
HTMAIL_ITEM_BLINDCOPYTO	BlindCopyTo
HTMAIL_ITEM_SUBJECT	Subject
HTMAIL_ITEM_BODY	Body
HTMAIL_ITEM_DELIVERYPRIORITY	DeliveryPriority
HTMAIL_ITEM_DELIVERYREPORT	DeliveryReport
HTMAIL_ITEM_RETURNRECEIPT	ReturnReceipt

htMailSend uses the following structure to easily submit common mail items and simple mail messages to the htMailSend function:

```
typedef struct
{
   char *sendto;              /* Separate names with semicolons
                              or commas */
   char *copyto;              /* Separate names with semicolons
                              or commas */
   char *blindcopyto;         /* Separate names with semicolons
                              or commas */
   char *subject;             /* Subject field */
   char *body_text;       /* If this is NULL, use body_comp */
   HTCOMPHANDLE body_comp;    /* If this is NULL, use body_text
*/
} HTMEMO;                 /* Simple mail memo structure */
```

The mail group contains the following function:

htMailSend	Sends a mail message

htMailSend

Summary Sends a mail message.

Syntax

```
HTSTATUS htMailSend (cursor, maildoc, memo, flags);
HTCURSOR  cursor;        /* Input */
HTDOCID   maildoc;       /* Input, Optional */
HTMEMO    *memo;         /* Input, Optional */
HTFLAGS   flags;         /* Input */
```

Description Creates and sends a mail message, using data from various sources. The contents of the message can come from an existing document, a C structure, bound items, or from any combination of these sources. Items in any existing document supersede all others, and items in the C structure supersede bound items (e.g., if the subject is in all three places, HiTest uses the value in the document). Every mail message must have at least one valid addressee in the SendTo item. The ability to sign or encrypt mail is unavailable since this functionality is not currently available in the standard Lotus Notes API.

Parameters

CURSOR

The cursor in which to base the message document. This is the source for the maildoc and bound items, if given. The HTMAIL_SAVE flag saves the mail document into this database.

MAILDOC

The document to send. HiTest adds data from the other sources to this document before sending the message. If there is no base document use NULLID and HiTest will create a new document. HiTest requires this parameter with the HTMAIL_EMBED_FORM flag. Embedding a form removes the Form item from the document, and embeds the form data itself within the document. Use form embedding when sending a document of a form which is not available in the recipient's mail database. Otherwise, when the recipient opens the received document, the form is unavailable and Notes cannot properly display the document.

MEMO

The C structure that contains standard memo items. HiTest adds these items to the message document. Using this structure and no other data sources is an easy way to send messages with the standard items. When using information from this structure to create a mail message with no base document (i.e., the maildoc parameter is NULL), the created document is of form "Memo". Programs must set all unused fields in this structure to NULL. See the Mail object section preceding the htMail functions for a description of this structure and its contents.

FLAGS

Zero or more HTMAIL flags OR-ed together. The HTMAIL_BOUND_ITEMS flag directs HiTest to insert bound items into the message (they are not added otherwise). The HTMAIL_EMBED_FORM flag directs HiTest to embed the mail document's form within the mail message (see maildoc parameter above). The HTMAIL_SAVE flag directs HiTest to save a copy of the mailed document in the supplied cursor's database. The HTMAIL_SAVE_MAILDB flag directs HiTest to save a copy of the mailed document in the Notes user's mail database. See the Mail object section preceding the htMail functions for a list and description of mail flags.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_DATA_UNAVAIL (no data to send or no SendTo value);

HTFAIL_INVALID_DATABASE (cannot find mail database to save message in);

HTFAIL_INVALID_FORM (cannot find form to embed);

HTFAIL_INVALID_DOCUMENT (embed form requires a valid maildoc).

Example

```
HTSTATUS status;
HTMEMO memo;
memset (&memo, 0, sizeof (HTMEMO));
memo.sendto = "David Letterman";
memo.subject = "Stupid Pet Tricks";
status = htMailSend (cursor, NULLID, &memo,
                                        HTMAIL_SAVE       |
HTMAIL_RETURN_RECEIPT);
```

See Also htCurOpen, htItemBind

Server

The server object has no context beneath the process level, but represents Lotus Notes servers as distinct objects. HiTest normally represents the local or NULL server by a NULL or empty string. Programs may assign an alternate value for the local server with the htSetOption function and HTGLOBOPT_LOCAL_SERVERNAME enumeration (e.g., the Lotus Notes UI would use the value "Local"). The primary attribute of a server is the server name.

Remote Notes databases are accessed through a server, which imposes Notes security restrictions on those databases. The server functions support access to basic server information and the ability to execute commands as if entering them at the Notes server.

The server group contains the following functions:

htServerExec	Remotely executes a Notes server console command
htServerGetInfo	Obtains a piece of information from and about a server
htServerList	Iterates through available Notes servers

htServerExec

Summary Remotely executes a Notes server console command.

Syntax

```
HTSTATUS htServerExec (server, command);
char      *server;        /* Input */
char      *command;            /* Input */
```

Description Executes a command against the indicated Notes server as if entering that command into the server's console. Use htServerGetInfo to retrieve the results of the most recent htServerExec call. Successful use of this function requires administrator access. See the Lotus Notes Administrator Guide for the syntax of server console commands. Two powerful console commands which lend themselves to remote use are REPLICATE (to initiate replication remotely) and LOAD (to load a program remotely).

Parameters

SERVER

The server on which to execute the command.

COMMAND

The command to execute on the server. See the Lotus Notes Administrator Guide for a list of valid server console commands.

Returns HTSTATUS return code. Failures include:

HTFAIL_DATA_UNAVAIL (the local server is invalid -- use a true server name).

Example

```
char *buffer;
HTINT length;
HTSTATUS status;
status = htServerExec ("MyServer", "SHOW TASKS");
if (!status)
{
                  status    =    htServGetInfo    (cursor,
HTSERVINFO_RESPLEN,
                              &length);
     buffer = malloc (length + 1);
```

```
                status = htServGetInfo (cursor,
HTSERVINFO_RESPSTR,
                                buffer);
}
```

See Also htServerList, htServerGetInfo

htServerGetInfo

Summary Obtains a piece of information from and about a server.

Syntax

```
HTSTATUS htServerGetInfo (server, item, buffer);
char      *server;       /* Input */
HTSERVINFO     item;          /* Input */
void      *buffer;       /* Output */
```

Description Fetches one of various server-level information items into a supplied buffer. Each item has a data type, and the buffer must be large enough to hold the result.

Parameters

SERVER

The server about which to obtain information.

ITEM

One value from an enumeration of server items. Each item corresponds to a type (and length, for variable length types). The following table lists legal items with their corresponding data types and, where relevant, lengths:

constant	type
HTSERVINFO_PING	HTBOOL
HTSERVINFO_RESPLEN	HTINT
HTSERVINFO_RESPSTR	char [HTSERVINFO_RESPLEN]

PING determines whether the server is available;

RESPLEN obtains the length of the server's response from the most recent htServerExec;

RESPSTR obtains the text of the server's response from the most recent htServerExec.

BUFFER

The buffer to receive the requested information. This buffer should be of sufficient length to handle the result.

Returns HTSTATUS return code. Failures include:

HTFAIL_DATA_UNAVAIL (current response results are not for the indicated server); HTFAIL_ILLEGAL_ENUM (invalid item).

Example

```
HTBOOL exists;
HTSTATUS status;
status = htServerGetInfo ("MyServer", HTSERVINFO_PING,
                          &exists);
```

See Also htServerList, htServerExec

htServerList

Summary Iterates through available Notes servers.

Syntax

```
HTSTATUS htServerList (operation, server);
HTLIST    operation;          /* Input */
char      *server;       /* Output */
```

Description Returns the first or next server name from the list of available Notes servers. The local server is normally not returned as a server. Assigning a local server name value with the htSetOption function will cause this function to return the local server string as the first server name in the server list.

Parameters

OPERATION

An element of the HTLIST enumeration that indicates whether and how to reset the server list. Use HTLIST_REFRESH to discard the server list and obtain a new list from Notes. Use HTLIST_FIRST to set the next element in the list to the first element. Use HTLIST_NEXT to obtain the element following the last fetched element. The first call to this function following htInit always uses the value HTLIST_REFRESH.

SERVER

A character buffer which receives the server name. The constant HTLEN_SERVERNAME defines the maximum server name length.

Returns HTSTATUS return code. Failures include:

HTFAIL_END_OF_DATA (no more servers).

Example

```
char server_name [HTLEN_SERVERNAME + 1];
HTLIST list_op = HTLIST_FIRST;
HTSTATUS status;
printf ("List of Notes Servers:");
while (!htServerList (list_op, server_name))
{
     printf ("\n      '%s' ", server_name);
     list_op = HTLIST_NEXT;
}
```

See Also htSetOption

TextList

Use text list functions to access the components of text list *items*. These functions provide simple interfaces to access the elements of a text list. See the “Data Types” section of Chapter 3, “Programming to the HiTest API” for a description of text list data.

The text list group contains the following functions:

htTextListCount	Returns the number of text elements in a text list
htTextListFetch	Copies a text list element into a supplied buffer
htTextListGetPtr	Returns a pointer to a text list element
htTextListLength	Returns the length of either one text list element, or the entire text list

htTextListCount

Summary Returns the number of text elements in a text list.

Syntax

```
HTSTATUS htTextListCount (textlist);
void      *textlist;          /* Input */
```

Description Returns the number of text elements in a text list.

Parameters TEXTLIST

A pointer to the text list data.

Returns HTINT number of text elements in the text list.

Example

```
HTINT count;
count = htTextListCount (textlist_ptr);
```

See Also htTextListLength, htTextListFetch, htTextListGetPtr

htTextListFetch

Summary Copies a text list element into a supplied buffer.

Syntax

```
HTINT htTextListFetch (textlist, index, buffer);
void      *textlist;          /* Input */
HTINT          index;         /* Input */
char      *buffer;       /* Output */
```

Description Copies the data for a specified text list element into a supplied buffer. To simply obtain a pointer to the data within the text list data itself, use the similar function htTextListGetPtr.

Parameters

TEXTLIST

A pointer to the text list data.

INDEX

The index of the text list element to copy into the supplied buffer. The first element is one. Use htTextListCount to obtain the last element's index.

BUFFER

The buffer into which to copy the text element. This buffer must be of sufficient length. Use htTextListLength to determine the length of a specific text list element.

Returns HTINT length of the text copied into the buffer. Returns zero if index is invalid.

Example

```
char *buffer;
HTINT length;
buffer = malloc (htTextListLength (textlist_ptr, 1) +
1);
length = htTextListFetch (textlist_ptr, 1, buffer);
```

See Also htTextListCount, htTextListLength, htTextListGetPtr

htTextListGetPtr

Summary Returns a pointer to a text list element.

Syntax

```
char *htTextListGetPtr (textlist, index, length);
void      *textlist;          /* Input */
HTINT          index;         /* Input */
HTINT          *length;       /* Output, Optional */
```

Description Returns the pointer to a specified text list element, and obtains the length. Modifying data at this pointer would modify the data in the text list buffer, which is not a valid action on a text list obtained with a GetPtr function. To copy the data into a supplied buffer, use the similar function htTextListFetch.

Parameters

TEXTLIST

A pointer to the text list data.

INDEX

The index of the text list element to obtain. The first element is one. Use htTextListCount to obtain the last element's index.

LENGTH

The buffer to receive the length of the text element at the pointer returned. HiTest sets this value to zero if the index is invalid.

Returns char * pointer to the text list element within the textlist data. Returns NULL if index is invalid.

Example

```
char *buffer;
HTINT length;
buffer = htTextListGetPtr (textlist_ptr, 1, &length);
```

See Also htTextListCount, htTextListLength, htTextListFetch

htTextListLength

Summary Returns the length of either one text list element, or the entire text list.

Syntax

```
HTINT htTextListLength (textlist, index);
void      *textlist;          /* Input */
HTINT          index;         /* Input, Optional */
```

Description Returns the length of either a single text list element, or the entire text list (depending on the element index).

Parameters

TEXTLIST

A pointer to the text list data.

INDEX

The index of the text list element whose length is to be determined. The first element is one. Use htTextListCount to obtain the last element's index. A value of zero determines the length of the entire text list.

Returns HTINT length of one element or the entire text list.

Example

```
HTINT length;
length = htTextListLength (textlist_ptr, 1);
```

See Also htTextListCount, htTextListFetch, htTextListGetPtr

View

A view is one of two primary types of metadata (the other is a *form*). Each database contains one or more views, which describe one representation of some or all of the documents in a database. A view consists of various attributes and one or more *columns*. In addition, each view defines an index. A view's index is a hierarchical, sorted collection of documents. A view also contains data, in the form of a table of *cells*. Lotus Notes represents views as simply data, and the standard Lotus Notes API supplies no abstraction of this data. The HiTest view abstraction supports easy access to view metadata. The primary attributes of a view are a name and ID. HiTest uses the constant NULLID to represent an invalid view ID.

The following flags define view attributes in the HTVIEW structure:

HTVIEW_COLLAPSED	Open view collapsed (default is expanded)
HTVIEW_NO_HIERARCHY	View is flat (no responses)
HTVIEW_DISP_ALL_UNREAD	Display unread markers in margin for all documents
HTVIEW_DISP_CONFLICT	Display replication conflicts
HTVIEW_DISP_MAIN_UNREAD	Display unread markers in margin only for main documents
HTVIEW_USES_TOTALS	One or more columns are totaled

htViewList returns the following view summary structure:

```
typedef struct
{
   HTVIEWID viewid;            /* View ID */
   HTBOOL hidden;                   /* Whether view is hidden in
UI */
   char name [HTLEN_DESIGNNAME + 1];               /* View name */
   char display_name1 [HTLEN_DISPLAYNAME + 1];     /*      Primary
display
                                              name */
   char display_name2 [HTLEN_DISPLAYNAME + 1];     /*    Secondary
display
                                              name */
} HTVIEWSUMM;                  /* htViewList summary structure */
```

htViewGetAttrib returns the following view attribute structure:

```
typedef struct
{
   HTFLAGS flags;                   /* View flags (HTVIEW_xxx) */
   WORD background_color;      /* Background color */
   HTFONT title_border_font;        /* Font for title and borders
*/
   HTFONT unread_font;              /* Font for unread rows */
   HTFONT totals_font;              /*   Font   for   totals   and
statistics */
   WORD update_interval;            /* Seconds between automatic
updates
```

```
                                    Use zero for no auto update */
   HTBOOL hidden;                    /* Whether view is hidden in
UI */
   char name [HTLEN_DESIGNNAME + 1];                /* View name */
   char display_name1 [HTLEN_DISPLAYNAME + 1];     /*      Primary
display
                                                   name */
   char display_name2 [HTLEN_DISPLAYNAME + 1];     /*    Secondary
display
                                                   name */
} HTVIEW;                                       /*     View     attribute
structure */
```

The three name fields in the HTVIEW and HTVIEWSUMM structures handle Notes' multiple naming of objects. Views may have multiple names, and the first name may consists of two parts. The *name* field contains the string which Notes uses internally to refer to a given view. The *display_name1* field contains the name which appears in the Notes UI. For a cascading view name, the *display_name2* field contains the cascading component of the Notes UI name. When a view has only one name, the *name* field is equal to either *display_name1* (if not cascading) or *display_name1*/*display_name2* (if cascading). The hidden field indicates whether Notes normally displays the view in the Notes UI View menu. A hidden view has its the display name enclosed in parenthesis.

The view group contains the following functions:

htViewCopy	Copies a view from one cursor to another
htViewDelete	Deletes a view from a database
htViewGetAttrib	Obtains the attributes of a view
htViewGetId	Obtains a view ID from the view name
htViewList	Iterates through views in a database
htViewSet	Assigns the active view for a cursor

htViewCopy

Summary Copies a view from one cursor to another.

Syntax

```
HTSTATUS    htViewCopy    (src_cursor,    src_viewid,
dest_cursor,
                    dest_viewname,  dest_viewid);
HTCURSOR  src_cursor;          /* Input */
HTVIEWID  src_viewid;          /* Input */
HTCURSOR  dest_cursor;         /* Input */
char      *dest_viewname;          /*  Input,  Optional
*/
VIEWID    *dest_viewid;        /* Output, Optional */
```

Description Copies a view between cursors, optionally assigning a new name. Notes requires view names within a database to be unique.

Parameters

SRC_CURSOR

The cursor from which to copy the view.

SRC_VIEWID

The view to copy within the source cursor.

DEST_CURSOR

The cursor into which to copy the new view.

DEST_VIEWNAME

The name for the new view in the destination cursor. To keep the original name, use NULL or the empty string. Otherwise, the name formatting follows the Notes UI rules (“display_name1\ display_name2 | name” -- see the Lotus Notes Application Developer’s Reference). A new name is required when the source and destination cursors are the same.

DEST_VIEWID

The buffer which receives the view ID for the new view.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_INVALID_VIEW (source view does not exist);

HTFAIL_DUPLICATE (a view exists in the destination cursor with the same title);

HTFAIL_OVERFLOW (new view title is too long).

Example

```
HTVIEWID new_viewid;
HTSTATUS status;
status = htViewCopy (cursor1, viewid, cursor2,
"NewView",
                    &new_viewid);
```

See Also htViewGetId, htViewDelete

htViewDelete

Summary Deletes a view from a database.

Syntax

```
HTSTATUS htViewDelete (cursor, viewid);
HTCURSOR  cursor;         /* Input */
HTVIEWID  viewid;         /* Input */
```

Description Deletes a view from a database. A cursor's active view cannot be deleted.

Parameters CURSOR

The cursor containing the view.

VIEWID

The view to delete.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_INVALID_VIEW (view does not exist);

HTFAIL_ACTIVE_RESULT (cannot delete the active view).

Example

```
HTSTATUS status;
status = htViewDelete (cursor, viewid);
```

See Also htViewGetId, htViewCopy, htViewSet

htViewGetAttrib

Summary Obtains the attributes of a view.

Syntax

```
HTSTATUS htViewGetAttrib (cursor, viewid, view);
HTCURSOR  cursor;         /* Input */
HTVIEWID  viewid;         /* Input */
HTVIEW    *view;          /* Output */
```

Description Obtains complete attributes for a view.

Parameters

CURSOR

The cursor containing the view.

VIEWID

The view for which to obtain attributes.

VIEW

The structure to receive view attributes. See the View object section preceding the htView functions for a description of this structure and its contents.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_INVALID_VIEW (view does not exist).

Example

```
HTVIEW view;
HTSTATUS status;
status = htViewGetAttrib (cursor, viewid, view);
```

See Also htViewList, htViewGetId

htViewGetId

Summary Obtains a view ID from the view name.

Syntax

```
HTVIEWID htViewGetId (HTCURSOR cursor, char *viewname);
HTCURSOR  cursor;         /* Input */
char      *viewname;          /* Input */
```

Description Given a view name, obtains the view ID of the indicated view.

Parameters

CURSOR

The cursor containing the view.

VIEWNAME

The view name for which to obtain the ID.

Returns HTVIEWID for the requested view. Returns NULLID if the view does not exist.

Example

```
HTVIEWID viewid;
viewid = htViewGetId (cursor, "All By Date");
```

See Also htViewList, htViewGetAttrib

htViewList

Summary Iterates through views in a database.

Syntax

```
HTSTATUS htViewList (cursor, operation, viewsumm);
HTCURSOR  cursor;        /* Input */
HTLIST    operation;          /* Input */
HTVIEWSUMM     *viewsumm;          /* Output */
```

Description Returns the first or next view information from the list of views within the cursor's database.

Parameters

CURSOR

The cursor from which to list views.

OPERATION

An element of the HTLIST enumeration that indicates whether and how to reset the view list. Use HTLIST_REFRESH to discard the view list and obtain a new list from Notes. Use HTLIST_FIRST to set the next element in the list to the first element. Use HTLIST_NEXT to obtain the element following the previously fetched element. The first call to this function after opening the cursor always uses the value HTLIST_REFRESH.

HTVIEWSUMM

The structure to receive the view's summary information. See the View object section preceding the htView functions for a description of this structure and its contents.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_END_OF_DATA (no more views).

Example

```
HTVIEWSUMM viewsumm;
HTSTATUS status;
status = htViewList (cursor, HTLIST_NEXT, &viewsumm);
```

See Also htViewGetAttrib

htViewSet

Summary Assigns the active view for a cursor.

Syntax

```
HTSTATUS htViewSet (HTCURSOR cursor, HTVIEWID viewid);
HTCURSOR  cursor;         /* Input */
HTVIEWID  viewid;         /* Input */
```

Description Sets the active view for a cursor. Producing a view-based index requires an active view. When there is an active view, calling htFormulaExec with no formula produces a hierarchical index from the view. Use the htCurGetInfo function to obtain a cursor's active view. This function is invalid in a cursor containing an active index.

Parameters

CURSOR

The cursor in which to set the active view.

VIEWID

The view to set as the active view. Use NULLID to clear the active view.

Returns HTSTATUS return code. Failures include:

HTFAIL_INVALID_CURSOR (invalid cursor);

HTFAIL_INVALID_VIEW (view does not exist);

HTFAIL_ACTIVE_RESULT (cannot set the active view with an active index).

Example

```
HTSTATUS status;
status = htViewSet (cursor, "All By Date")
```

See Also htViewGetId, htCurSetOption, htCurGetInfo, htFormulaExec, htCellBind

Glossary

This section defines some of the Notes-related terms used throughout this document. In addition, some of the terminology in this document relates to database APIs in general, and some familiarity in this area is helpful, although not required. Some of these terms are also described below.

API — API stands for Application Programming Interface, a set of functions and supporting code and documentation which provides a clean interface to some application. HiTest is a high-level API to Lotus Notes.

attachment — A file not contained within a document, but bound to a document. Any document may have any number of attachments. Although the attachment is not stored in the document, Notes stores it in the document's database.

bound dataspace — A set of memory locations related to a set of specified document items and/or view cells. Using HiTest, data can be fetched to or inserted or updated from these memory locations to the assigned document items and view cells by a single function call.

category — A row within a view which represents a grouping of documents with some commonality. When a column is categorized, Notes groups all documents with the same value for that column together. Notes produces an extra non-document row for each grouping, and the cells in the categorized column for those rows contains the category value. The cells for the categorized column are empty for all other rows.

child — See response.

cell — A data value within a view-based index. Each column and row combination represents a cell. Cell contents cannot be modified -- they exist within Notes as read-only values.

column — A single piece of metadata within a view. HiTest describes a column by its integer location (e.g., column 1, column 2, ...), as well as other attributes. The data value within a column for a single view row is a cell.

composite — A Lotus Notes data type constructed from some number of smaller components called composite records. Each composite record may be in any of a set of data formats (e.g., bitmap, formatted text, audio). Also called rich text or compound text.

composite record — A component of a composite. One or more composite records make up a composite item. Each composite record is of a specific type (e.g., formatted text, graphic, doclink, etc.).

compound text — See composite data.

context — The conditions under which an action or object is valid. Certain actions and objects are only valid within the context of another object. For example, a composite handle is only valid within the context of its containing document handle, which in turn is only valid within its containing cursor. Closing the cursor closes and invalidates the document and composite handles.

cursor A handle which indicates a single API session. Programs may open multiple cursors. A cursor contains the following components: a Notes database handle; options; a formula buffer; an active form, view, and index; and open documents and composites.

database A collection of documents and metadata within a single .NSF file. Each cursor connects to exactly one database, although multiple cursors may simultaneously connect to the same database.

document A collection of data items. A document is how Notes stores data. Data within a database is stored in documents.

fetch The process of retrieving data. When fetching a document, a single function call retrieves and converts multiple bound items and cells.

field A single piece of metadata within a form. A field consists of a data type, a name, and other attributes. When using strict binding, all items within a document must also exist within the document's form as fields.

file attachment See attachment.

formula A statement executed against a database to perform a specific action. Currently, only selection formulas (i.e., those which produce an index) are valid through the HiTest API. The Lotus Notes formula language defines the syntax of a formula.

formula buffer A buffer used for constructing formulas for execution. Each cursor contains one formula buffer. After construction, the formula in the formula buffer may be executed.

formula language The grammar specification which defines the valid syntax for formulas. The Lotus Notes application documentation defines the Lotus Notes formula language syntax.

form A form is the type of metadata which defines the format for the creation or interpretation of documents. A document often, but not always, contains an item which indicates the document's form. While documents do not require a form (see strict binding), they should unless there is a good reason not to be based on a form. A single database may contain multiple forms.

full text search A search of all text within a set of documents. Lotus Notes has the ability to index a database for full text search. Programs may execute a full text search query against some or all of the documents within that database, and any documents which match the query are selected. A full text search query normally consists of one or more words or phrases. The Lotus Notes application documentation defines the Lotus Notes full text search query syntax.

handle A handle is a simple (usually integer) value used to indicate some context within the API. Operations which create open data objects (e.g., opening a cursor or document) produce a handle. An open object's handle indicates a particular open object to the API when multiple instances of that type of object may exist. For example, when multiple documents are open, the integer handle indicates which open document to use for a given operation.

hierarchical A tree-based set. In HiTest, view-based indices are hierarchical since each document in them may have a parent document and/or child documents.

HiTest The name of the enhanced Notes API described in this documentation.

index A set of documents produced by executing a selection formula within a cursor. The primary attribute of any index is an ordered collection of documents. There are two types of indices: flat and hierarchical (view-based). Flat indices are produced by a full database search or full text query, and have no hierarchy. Hierarchical indices are produced from the set of documents within a view, and contain a set of cells corresponding to the view display for each document. The process of moving through the documents within an index is navigation. Result set is another term for index. A cursor in which a formula has been executed and results are available contains an active index.

insert The process of creating a new document in a Notes database. Programs usually do this with bound dataspace or by creating an empty document and adding items one at a time.

item A data value within a document, on which the HiTest API supports retrieval and assignment. When strict binding is in effect, every item in a document must correspond in name and data type to a field in that document's form.

macro A stored Notes object which performs a specified action on request or on schedule. Macro execution can create, modify, or select documents, depending on the type of macro.

mail A message sent between e-mail users. Notes supports addressing and sending of documents as mail.

metadata Data which describes data. The types of metadata relevant to the API (listed with their corresponding data object) are:

forms (corresponding to documents);
fields within forms (corresponding to items within documents);
views (corresponding to the set of documents within views);
columns (corresponding to cells within views).

navigation The process of moving through an index. From any point within an index, there are various navigation styles (e.g., next, first, previous parent, etc.) usable to move to other documents in the index. View-based indices support a greater range of navigation styles than flat indices.

null A special undefined value for any data type. This is different from zero-values such as numeric zero and the empty string. In Notes documents, HiTest represents a NULL value by the absence of an item within a document, since Notes has no concept of a NULL value as data.

parent A parent document is a document to which another document is a response (child). One parent may have multiple responses. The only way to determine a document's responses is by view-based navigation.

query A statement executed against a database to perform an action. In Notes, a query is used to perform a full text search of some or all text within a database. The Lotus Notes application documentation defines the syntax of a query.

response A document which references another document as its parent. Programs can determine a document's parent either by view-based navigation or by the value of the document's reference item. The term response is synonymous with child.

response hierarchy The set and organization of responses beneath a given document is that document's response hierarchy, and may consist of multiple levels (responses may have responses). While a document may have many responses, it may only have one parent.

rich text See composite data.

row A single horizontal entry within a view. Each row (document, category, or totals) in a view contains one cell for each column in the view.

selection formula A formula which produces an index, which consists of a subset of the documents within a database.

server A named Notes server program that receives and processes requests from clients, contains and maintains a set of Notes databases, and implements security on the databases it controls.

session One API connection, with its own internal state and data. A given process may open multiple sessions, each of which may perform independent functions at the same time. Each session is indicated by a cursor.

state Each process has a state which defines valid operations at the current time (i.e., data is not accessible from a document which is not open). Certain operations within one state result in a different state.

strict binding There is no requirement in Notes that items within a document match the fields within that document's form, or even that a document have a form. Since this may cause confusion, and normally the ability to have documents differ in format from forms is unnecessary, the API enforces strict binding by default. This filters document access through the document's form, and forces a consistent structure on items in documents. Strict binding is a cursor-level option.

summary item Notes stores certain items as summary items, which may be used in computations and are usually smaller than other non-summary items. Composite items cannot be summary items. In general, most other items are summary items (up to the Notes limit of 15 K of summary data per document).

update The process of modifying an existing document in a Notes database. Programs perform updates by using bound dataspace or by adding or replacing items one at a time.

view A view contains a sorted and indexed hierarchical set of documents within a database. The set of documents within a view are accessible as a view-based index. A view also contains metadata in the form of columns and data in the form of cells.

Index

Instead of the conventional back-of-the-book tabular index, this document is also supplied as a Notes database that can be full text searched. Use the Notes UI to build a full text search index. Then use full text search to produce an interactive index with the additional search capabilities (relevance scores, sorting, etc.). We believe that a searchable documentation database coupled with a complete table of contents is the most effective indexing methodology for a structured manual such as this one.