



**[www.isys-search.com](http://www.isys-search.com)**

ISYS Search Software, Inc.  
Denver Technology Center  
8775 E. Orchard Rd, Suite 811  
Englewood, CO 80111 USA  
Tel: (303) 689 9998  
Fax: (303) 689 9997

ISYS Search Software Pty. Ltd.  
Suite 102, 10-12 Clarke St  
Crows Nest, NSW 2065  
Australia  
Tel: + 61 2 9439 5800  
Fax: + 61 2 9439 8569

ISYS Search Software (UK) Ltd.  
The Steam Mill  
Steam Mill Street  
Chester CH3 5AN  
Tel: + 44 1244 313 216  
Fax: + 44 1244 313 003

Documentation Version 8.00

---

## Commercial in Confidence

Copyright ©1988 - 2006 ISYS Search Software Pty Ltd.

While ISYS Search Software Pty Ltd. makes every effort to ensure that this manual is accurate and complete, ISYS Search Software Pty Ltd. denies liability for damages arising from the use of this product.

Information in this manual is subject to change without notice and does not represent a commitment on the part of ISYS Search Software Pty Ltd. The software described in this document is furnished under a license agreement or non-disclosure agreement. The software may be copied or used only in accordance with the terms of the agreement.

Trademarks appearing in this manual are the property of their respective owners.



---

# CONTENTS

<b>1.</b>	<b>Overview.....</b>	<b>13</b>
1.1	Installation.....	13
1.2	Getting Started.....	14
1.3	New in Version 8.0 .....	15
1.4	COM Interface .....	18
1.5	Multi-user metering .....	18
1.6	Distributing Completed Applications.....	19
1.7	Indexing API.....	20
1.8	Retrieval API.....	21
1.9	WYSIWYG API .....	21
1.10	Character Sets .....	22
1.11	Dealing with messages in Visual Basic.....	23
1.12	Using ISYS with non-English languages.....	24
1.13	Licensing and Support.....	24
<b>2.</b>	<b>Concepts .....</b>	<b>25</b>
2.1	Data Types and Parameter Passing.....	26
	Passing Parameters .....	28
2.2	Result Blocks .....	28
2.3	The Sample Applications .....	29
2.4	License Codes .....	30
<b>3.</b>	<b>Basic Retrieval.....</b>	<b>31</b>
3.1	Procedure Init_Instance .....	31
	Instance_Status_Block.....	32
3.2	Procedure Open_Database.....	34
3.3	Procedure Get_Database_Name .....	34
3.4	Procedure Perform_Find.....	35
	Query_Result_Block .....	36

3.5	Procedure Perform_English_Find .....	37
3.6	Procedure Sort_Documents .....	38
3.7	Procedure Get_Document_Record .....	40
	Document_Control_Block .....	41
3.8	Function Open_Document.....	44
3.9	Function Get_Line_With_Hit_After .....	45
3.10	Function Get_Line_With_Hit_Before .....	45
3.11	Procedure Get_Document_Line .....	47
3.12	Procedure Close_Document .....	51
3.13	Procedure Close_Instance .....	51
4.	Advanced Retrieval .....	53
4.1	Procedure ExistOnPath .....	54
4.2	Procedure FileName_Filter .....	54
	Filter_Block .....	55
4.3	Procedure Set_Sub_Find .....	57
4.4	Procedure Set_Rlist_Curfew .....	58
4.5	Procedure Set_Maximum_Found_Documents .....	58
4.6	Procedure Word_Search.....	59
	Word_Search_Context.....	59
4.7	Procedure Remove_Document_Record.....	60
4.8	Procedure Interrupt_Processing.....	60
4.9	Procedure Get_Hit_Pointer.....	60
	Word_Pointer_Block .....	61
4.10	Procedure Get_Document_Type .....	61
4.11	Function Get_Document_Fully_Qualified_FileName .....	61
4.12	Function Get_Document_Long_FileName .....	61
4.13	Procedure Get_Format_Attribute.....	61
4.14	Function Get_Document_Time_Stamp .....	62
4.15	Function Get_Document_Line_Page_No .....	62
4.16	Procedure Get_EAM_Context_Info_for_Hit .....	63

	EAM_Context .....	63
4.17	Function Auto_Determine_File_Format.....	64
4.18	Procedure Translate_Hyperstring .....	64
4.19	Function Open_Unfound_Document .....	65
4.20	Procedure Get_Term_ID_Term .....	66
4.21	Function Get_Line_With_Term_After .....	66
4.22	Function Get_Line_With_Term_Before .....	66
4.23	Procedure Get_Category_Record .....	67
	Category_Control_Block .....	67
4.24	Procedure Find_Similar .....	67
4.25	Procedure Get_Document_Group_Entry.....	68
4.26	Procedure Get_Query_Spelling_Suggestion .....	69
4.27	Procedure Get_Document_Metadata .....	69
4.28	Procedure Set_Metadata_Relevance_Boost.....	70
4.29	Procedure Set_Metadata_Query_Scope .....	70
4.30	Procedure Set_Metadata_Sort_Key .....	71
4.31	Procedure Close_Find.....	71
4.32	Procedure Close_Database .....	72
4.33	Performing Fielded Searching.....	72
5.	Utilities.....	75
5.1	Procedure CFG_Settings .....	76
	Configuration_Control_Block .....	76
5.2	Procedure CFG_Formats .....	79
	Format_Rec.....	79
5.3	Procedure Def_Settings .....	80
	Action_Control_Block.....	80
5.4	Procedure Open_Rules .....	81
5.5	Procedure Fetch_Rule_Areas.....	82
5.6	Procedure Fetch_Rule_List .....	82
5.7	Procedure Create_Rule .....	83

5.8	Procedure Move_Rule .....	83
5.9	Procedure Close_Rules .....	83
6.	Index Utilities .....	85
6.1	Procedure IDB_Function .....	85
	Constants for ISYS_MSG_Scanprogress .....	88
6.2	IDB_Function CREATE .....	88
6.3	IDB_Function UPDATE .....	88
6.4	IDB_Function OPTIMIZE .....	89
6.5	IDB_Function REINDEX .....	89
6.6	IDB_Function PREVIEW.....	89
6.7	IDB_Function STATISTICS .....	89
6.8	IDB_Function FREQ .....	90
6.9	IDB_Function LIST .....	90
6.10	IDB_Function COMMON.....	90
6.11	IDB_Function VERSION .....	91
6.12	IDB_Function USERS.....	91
6.13	IDB_Function SUPPORT .....	91
6.14	IDB_Function ADD .....	91
6.15	IDB_Function RECATEGORIZE.....	92
6.16	IDB_Function PROCESSCACHEDDELETES.....	92
6.17	IDB_Function REFRESHSECURITYCACHE .....	92
7.	Low-Level Indexing .....	93
7.1	Procedure Start_Concording_Run .....	95
7.2	Procedure Create_Database.....	95
7.3	Function Concord_From_File .....	95
7.4	Procedure Close_Concording_Run.....	96
7.5	Function DeConcord_From_File.....	97
7.6	Procedure Complete_DeConcording.....	97
7.7	Procedure Set_Concord_From_File_Option.....	98
7.8	Efficiency.....	99

<b>8.</b>	<b>Transactional Indexing.....</b>	<b>101</b>
8.1	Function Process_Transactions .....	101
8.2	Transaction File Format.....	102
8.3	Header.....	103
8.4	Confirm Document Exists (type 1) .....	103
8.5	Document No Longer Exists (type 2) .....	104
8.6	New or Changed Document (type 3) .....	105
8.7	Indirect New or Changed Document (type 6) .....	107
8.8	Rename Document (type 7).....	108
8.9	Non-Unicode Code Page (type 8) .....	109
<b>9.</b>	<b>External Access Modules.....</b>	<b>111</b>
9.1	Script EAM Alternative .....	112
9.2	Naming and Loading your EAM.....	112
9.3	Determining What Gets Indexed .....	113
9.4	Accessing the Documents.....	114
9.5	External Indexing.....	114
9.6	External Access Module API Routines .....	115
	File-Access Routines .....	115
	File-Directory Routines .....	116
	External-Filesystem Routines .....	116
	Additional External Access Module Routines.....	117
9.7	Function Ext_Scan_First_Directory.....	117
9.8	Function Ext_Scan_Next_Directory.....	117
9.9	Function Ext_Request_Document .....	118
9.10	Procedure Ext_Release_Document .....	118
9.11	Procedure Ext_Foreign_File_System .....	119
9.12	Function Ext_Open_Document .....	119
9.13	Function Ext_Get_Time_Stamp.....	120
9.14	Function Ext_Get_File_Size.....	120
9.15	Function Ext_Read_Character .....	121

9.16	Procedure Ext_Seek.....	123
9.17	Procedure Ext_Close_Document.....	123
9.18	Ext_Initialize.....	123
9.19	Ext_Finalize.....	124
9.20	Ext_Prepare_For_WEP.....	124
9.21	Procedure Ext_Get_Names .....	125
9.22	Procedure Ext_Auto_Format.....	125
9.23	Ext_Select_Output_Coding .....	126
9.24	EAMs and IFilters .....	126
10.	Installable Security Filters .....	127
10.1	Auth_Initialize .....	128
10.2	Auth_Uninitialize .....	128
10.3	Auth_Activate .....	128
10.4	Auth_Deactivate .....	129
10.5	Auth_Openable .....	129
10.6	Auth_GetLastError .....	131
10.7	Auth_GetSystemName .....	131
10.8	ISYS Auth Strings.....	131
10.9	User Contexts .....	132
10.10	Using Security Filters with ISYS:web.....	132
11.	Advanced Integration.....	133
11.1	Integration Routines.....	133
11.2	Function Direct_IXC_Read .....	133
11.3	Function Direct_IXC_Write .....	133
	Chapter_Entry_Block.....	134
11.4	Function Lookup_Document_By_Name.....	135
11.5	Function Rename_Indexed_Document .....	135
11.6	Function Get_Document_Id.....	136
11.7	Procedure Get_Chained_Database_List .....	136
	Concat_Block.....	136



11.8	Procedure ISYS_Multiplex .....	137
11.9	Procedure Switch_Instance .....	147
11.10	Procedure Set_Indexing_Callback_Hook.....	148
11.11	Procedure <i>MyIndexingCallback</i> .....	149
11.12	Procedure Set_Indexing_Filter_Hook.....	149
11.13	Function <i>MyFilteringCallback</i> .....	150
11.14	Procedure Attach_Custom_Relevance_Rating .....	150
11.15	Function <i>MyRelevanceRanker</i> .....	150
11.16	Procedure Merge_Index .....	151
11.17	Procedure Set_API_Charset.....	151
11.18	Procedure Widechar_to_UTF8 .....	152
11.19	Procedure UTF8_to_Widechar .....	152
11.20	Procedure Get_Document_Security_Descriptor .....	153
11.21	Procedure Get_Num_Processors .....	153
11.22	Procedure Set_Num_Indexing_Processes.....	154
11.23	Procedure Set_Engine_Call_Tracing.....	154
11.24	Using ISYS with other languages .....	155
11.25	Combining ISYS results with other sources .....	156
12.	Result-List Manipulators .....	157
12.1	Rlist Data Structure .....	157
12.2	Allocation and Deallocation.....	157
12.3	Rlist Manipulations .....	157
12.4	Rlist API Calls .....	159
12.5	Function Rlist_Create.....	159
	Result_List .....	160
12.6	Procedure Rlist_Drop .....	160
12.7	Procedure Rlist_Get_Entry .....	160
	Word_Pointer_Block .....	161
12.8	Procedure Rlist_Append_Entry.....	161
12.9	Procedure Rlist_Get_QRB .....	161
12.10	Procedure Rlist_Empty .....	162

12.11	Function Rlist_Current_Rlist.....	162
12.12	Procedure Rlist_Oper .....	163
12.13	Procedure Rlist_Copy_From_To.....	164
12.14	Procedure Rlist_Exchange.....	164
13.	Annotations .....	165
13.1	Procedure Get_Annotation_Index_Page.....	167
	Annotation_Entry .....	167
13.2	Procedure Get_Annotation_Entry.....	168
	Annotation_Control_Block.....	168
13.3	Procedure Set_Annotation_Entry .....	169
13.4	Procedure Get_Annotation_Entry_Text .....	170
13.5	Procedure Set_Annotation_Entry_Text.....	170
13.6	Procedure Write_Back_Annotations .....	171
13.7	Function Get_Annotations_Changed_Status .....	171
14.	ISYS Concept Taxonomies .....	173
14.1	Function SCT_Open .....	174
14.2	Procedure SCT_Entry .....	174
	SCT_Block.....	175
14.3	Procedure SCT_Replace.....	175
14.4	Procedure SCT_Insert_After .....	175
14.5	Procedure SCT_Move .....	176
14.6	Procedure SCT_Close.....	176
14.7	Procedure SCT_Close_and_Save .....	176
14.8	Procedure Perform_SCT_Find .....	177
14.9	Procedure SCT_Set_File_Name .....	178
15.	Synonyms .....	179
	SYN_Block .....	179
15.2	Function SYN_Get_Head .....	180
15.3	Procedure SYN_Get_Entry .....	180
15.4	Procedure SYN_Replace_Word .....	180

15.5	Procedure SYN_Delete_Entry.....	181
15.6	Procedure SYN_Add_Word_To_Ring .....	181
15.7	Procedure SYN_Undo .....	181
15.8	Procedure SYN_Save .....	181
16.	Named Sections .....	183
16.1	Procedure Field_Get.....	183
16.2	Procedure Field_Set .....	184
16.3	Procedure Field_Save .....	184
16.4	Procedure Field_Undo .....	184
17.	WYSIWYG Documents .....	185
17.1	Overview .....	185
17.2	Licensing .....	185
17.3	Creating WYSIWYG Documents .....	186
17.4	Opening a WYSIWYG Document.....	186
17.5	The WYSIWYG_Multiplex call .....	187
17.6	Getting the Visible hWnd .....	187
17.7	Handling Messages .....	189
17.8	Displaying the First Hit.....	190
17.9	Determining Navigability.....	191
17.10	Navigating .....	192
17.11	Printing .....	192
17.12	Determining how much of the document has been read.....	193
17.13	Determining the visible location .....	193
17.14	Determining the Page Number .....	194
17.15	Opening in non-WYSIWYG mode.....	194
17.16	WYSIWYG to non-WYSIWYG Function Relationship Chart.....	195
17.17	Other KeyView Features .....	196
17.18	Distributing WYSIWYG Applications .....	197
18.	Intelligent Agent.....	199
18.1	Procedure IA_Load .....	199
18.2	Procedure IA_Save .....	200

18.3	Procedure IA_Close .....	200
18.4	Procedure IA_Count.....	200
18.5	Procedure IA_Edit .....	201
18.6	Function IA_Evaluate .....	202
18.7	Function IA_Store_Size .....	202
18.8	Function IA_Activate.....	203
18.9	Function IA_ListForDel .....	204
18.10	Procedure IA_Save_Store_With_Dels .....	204
18.11	Procedure IA_Get_Next_Due_Agent .....	205
19.	Entity Recognition.....	207
19.1	Entity Names.....	207
19.2	Procedure Get_Entity_Summary.....	208
	Entity_Summary_Block .....	209
19.3	Procedure Get_Document_Entity_Summary .....	209
19.4	Procedure Get_Document_Entity_List.....	210
	Entity_Occurrence_Block.....	210
19.5	Procedure Get_Document_Entity_Record .....	211
19.6	Procedure Get_Entity_Type_Name.....	211
19.7	Function Get_Line_With_Entity_After.....	212
20.	COM Interface .....	213
	Appendix C: Document Format Codes .....	215
	Appendix D: Constants .....	217
	Appendix E: Data Structures .....	229

---

# 1. OVERVIEW

Under Windows, the core ISYS text indexing and search engine is implemented as a Dynamic Link Library (DLL). This manual describes the Applications Programming Interface (API) to the ISYS Text indexing and search Engine. The engine is the same one that ISYS Search Software has been using in its range of products since 1988, and is hence mature. ISYS Search Software uses this API itself to create its retail versions of its various ISYS products, including ISYS:desktop and ISYS:web. ISYS Search Software continually enhances and refines the ISYS engine to keep it at the forefront of knowledge management and search engine technology.

A COM interface to the engine is also provided, and is documented in a help file provided with the SDK.

This manual is intended for application programmers who have some experience in one programming language but not necessarily been exposed to others (for instance, a Visual Basic programmer with little or no C experience). This manual assumes you know how to use your programming tool; it will not attempt to teach you how to use it.

This manual contains an overview of the ISYS Engine and its relationship to documents, indexes, and applications. The basic API calls are described first, followed by more powerful and more specialized groups of calls. You only need to read the first chapter to write a basic ISYS query tool; you can look at subsequent chapters for additional feature needs.

## 1.1 Installation

The first step is to install the retail version of ISYS:desktop that was supplied with the SDK. Instructions for doing so are included in the Quick Start manual provided with ISYS:desktop. Be sure that the licensee name you enter during the install is correct and appropriate.

You are not licensed to redistribute ISYS:desktop, but ISYS:desktop is provided with the SDK because it gives you a simple and effective way to get started with your own indexes, and a good way of comparing your own application against another application written to the same API.

Having installed ISYS:desktop, now make a folder for the SDK materials and copy the entire contents on the SDK CD into it.

You should check the README.TXT file for any additional information.

## 1.2 Getting Started

The easiest way to get started is to configure and build an ISYS index using ISYS:desktop as provided. You can then perform queries using ISYS:desktop to ensure that you understand the core functionality of the underlying ISYS engine.

Then, using the programming language of your choice, code a small application that:

- Calls Init\_Instance
- Calls Open\_Database
- Calls Perform\_Find
- Displays the contents of the Query Result Block
- Calls Close\_Database
- Calls Close\_Instance

The next step of sophistication is to make some calls to Get\_Document\_Record to display the names of the documents found and the number of hits in each. The next step after that is to call Open\_Document and Get\_Document\_Line to browse the found documents, or make use of the WYSIWYG API.

At each step, compare the results you see with the results obtained from ISYS:desktop. ISYS:desktop is just an application that uses the same DLL you are calling, so anything that ISYS:desktop does, you can achieve in your application as well.

If you are intending to use the COM interface, after chapters one and two, proceed directly to the ISYS COM manual.

Sample code and header definitions are provided for a variety of languages in a variety of subdirectories on this disk. Simply access the sample source you need.

<b>DELPHI</b>	<b>Borland Delphi</b>
<b>VB</b>	<b>Microsoft Visual Basic</b>
<b>C</b>	<b>Microsoft C</b>
<b>ASP</b>	<b>ASP pages</b>

There are also subdirectories containing sample EAMs in C and Delphi, and sample use of the COM object in VB and Delphi, as well as ASP and Cold Fusion.

Different languages require different techniques for integrating the ISYS engine into your applications.

<u>Language</u>	<u>How to Integrate ISYS with Your Application</u>
Visual Basic	Add ISYS8.BAS to your project. It contains DECLARE and TYPE statements that enable your program to talk to the ISYS Engine.
C	Include ISYS8.H which dynamically binds to the ISYS engine without the need for a LIB file to link against.
Borland Delphi	Include ISYS8.PAS in a Uses statement and compile and run your program in the usual way.
Other Languages	You will need to use this manual and/or the above files to help construct procedure definitions to be used with your language.

Note that these samples are for calling the ISYS DLL directly. Information on calling the engine via a COM interface is available in the ISYS COM manual.

## 1.3 New in Version 8.0

The V8 API is upwardly compatible with the V7 API. Some data structures have changed, but the C, Delphi, and Visual Basic interface files have been updated to reflect the changes. Most programs should only require a recompile to implement the changes.

The name of the DLL being called has changed to **ISYS8.DLL**, thereby allowing version 7 API programs to co-exist with version 8 API programs.

Developers who are upgrading directly from version 6 to version 8 should obtain a copy of the version 7 SDK manual and carefully read the section “New in Version 7.0”, because there were substantial changes to the licensing scheme and character representation conventions implemented between those two versions.

**CFG\_Settings** data structure has been expanded to include:

- Increased length for significant and insignificant character items.
- Added a **Index\_Type** field to identify indexes which have been configured specifically for ISYS:spider or ISYS:email use.
- Added a **Custom\_Props** field to allow customized properties to be stored in the ISYS.CFG file and accessed for application specific purposes.
- Added a **Entity\_Handling** field to control whether intelligent entity recognition occurs at indexing time.
- Added a **Cache\_Documents** field to control whether document textual content is extracted at indexing time and cached in compressed form into the index to allow higher performance access at query time.

The **Document\_Control\_Block** data structure has been expanded to include:

- Added a **Paragraph\_Count** field to provide access to the number of paragraphs in the document.
- Added a **Flags** field which contains bitfield information about the existence of associated documents which may be later accessed in detail by calling **Get\_Document\_Group\_Entry**.
- Added an **Entity\_Count** field which indicates the number entities recognized in this document.

The **Filter\_Block** data structure has been expanded to include a file type field to allow filtering based on file format.

Added a **Set\_Synonym\_Callback\_Hook** call to allow SDK users to insert their own synonym expansion engine.

Added a series of APIs to externalize entity recognition functionality, which automatically recognizes and summarizes the “who, what and where” involved in a result list. New entry points are **Get\_Entity\_Summary** to access a summarized entity list for a query result, **Get\_Document\_Entity\_Summary** to access a summarized entity list for a particular document, **Get\_Document\_Entity\_List** to access a detailed entity list for a particular document, **Get\_Document\_Entity\_Record** to access a details of a particular entity, **Get\_Line\_With\_Entity\_After** to enable entity-to-entity navigation within a document, and **Get\_Entity\_Type\_Name** to map entity type codes to entity type names.



Added new **ISYS\_Sort\_Cat\_Weighted\_Relv** sort method to sort by relevance weighted by the relevance of the category to which the document belongs.

Added new **ISYS\_Query\_AND\_With\_Current** option which may be applied to the search entry points and which causes the query to be ANDed with the current result set, and with highlighting provided for both sets of results. This may sometimes be used in preference to **Set\_Sub\_Find**.

Added new **BODYONLY** and **METAONLY** query syntax which may be used to limit the subexpression which follows to the nominated portion of the document, for example:

METAONLY CAT\*

CAT AND METAONLY (FLEA OR TAIL) and BODYONLY (CAT // DOG)

Added new **FNAMELIKE**, **FNAMEUNLIKE**, **PATHCONTAINS**, **PATHOMITS**, **CATEGORYLIKE**, **CATEGORYUNLIKE**, **PATHOMITS**, **FILEDATEBEFORE**, **FILEDATEAFTER**, **INDEXEDBEFORE**, **INDEXEDAFTER**, **FIRSTDATEAFTER** and **FIRSTDATEBEFORE** query syntax which allow file name filters to be specified as part of the query syntax, instead of using the **FileName\_Filter** API. These filters are global to the entire query command, and must be followed by a single quoted parameter. For example:

CAT FNAMELIKE "\*.DOC"

PATHOMITS "BROCHURES" CAT

PATHCONTAINS "ABC;DEF;GHI" FNAMEUNLIKE "\*.BAK" CAT

\* INDEXEDAFTER "20060323"

Added new **ResetLastAccessDates** keyword for the ISYS.CFG file which causes ISYS to cache and replace the Windows "last accessed date" for each file opened during indexing, and replace the date after ISYS has finished indexing the file.

Added support for **WMA**, **WMV** and **ASF** files, as well as **XPS** (XML paper specification) and **Microsoft Office 2007** (as at the time of publication, tested against the Office 2007 beta).

**ISYS\_Multiplex** has been extended with many additional options, including various functions to return additional information about the index and the environment.

Various **limits have been increased**, including:

- Maximum amount of cached metadata per document is now 6kb (you can have up to 255 metadata fields, all searchable and accessible, but only the first 6k will be cached for rapid access).
- Maximum number of lexical tokens per query has been increased to 1024.
- Maximum number of indexes in a single query chain remains defaulted to 128, but may be increased to any arbitrary limit using **IndexChainLimit**.

**Chapter\_Entry\_Block** as used in `Direct_IXC_Read` and `Direct_IXC_Write` has been extended.

When configured with the same indexing options enabled, version 8 builds indexes around 10% **faster** than version 7.

A **Linux** version of the SDK is now available which includes all the main features of ISYS for Windows, and with an API which is almost identical. Contact ISYS Search Software for more details.

## 1.4 COM Interface

This manual describes how to call the ISYS engine directly, which is the most efficient way of interfacing to the engine and affords the most control. If you are planning to call the ISYS engine via the supplied COM wrapper, please see the documentation supplied alongside this manual.

## 1.5 Multi-user metering

ISYS may operate in one of two modes, depending on your licensing arrangements with ISYS Search Software.

If your application is for commercial resale, you will have been provided with a license code which allows unlimited usage of ISYS, and any form of multi-user control or metering is up to you.

If your application is for your own internal use, you may have been provided with a license code which only permits a limited number of seats to run. In this case, you will need to run the ISYS License Code Manager (ILCM.EXE) to create your ISYS8.LIC file and to nominate a shared subdirectory which will be used for license control purposes.

If you are evaluating the ISYS SDK, you will have been provided with an evaluation license code which allows any number of users to run, but which may display a message to indicate an evaluation is under way.

## 1.6 Distributing Completed Applications

When you have completed your application and are ready to distribute it, there are a number of deployment considerations.

Your application would normally install into its own directory, and place all its executables in that directory. We recommend you place your ISYS engine modules in the same directory. At a minimum, these are:

- Your executable program (filename.EXE),
- The ISYS engine, ISYS8.DLL
- Your ISYS8.KEY file

When your application first calls the ISYS engine, it tells ISYS the base executable path from where ISYS will expect to find its other modules. All directory references relate to that executable directory.

Your application must pass a valid license code. See the `Init_Instance` call for further details.

Your installation of ISYS:desktop serves as a sample of how to install an application which uses the ISYS engine.

If your application is for your internal use, you will also need to run the ISYS License Code Manager (ILCM.EXE) to create your ISYS8.LIC file and nominate a directory for multi-user control. This will ensure you do not exceed your licensed user capacity.

If your application uses the email indexing or Rich HTML features, you will also have to redistribute ISYSU8.DLL, which should be placed alongside the ISYS8.DLL module.

If your application makes use of the SQL data indexing feature of ISYS, you will also need to redistribute the contents of the SQL directory, which should be placed beneath the executable directory, and the ISYSSQL.DLL file, which should be placed alongside your ISYS8.DLL.

If your application makes use of the built-in thesaurus feature of ISYS, you will also need to redistribute the contents of the WORDNET directory, which should be placed beneath the executable directory, and also ISYSWNET.DLL which should be placed in the executable directory.

If your application makes use of PDF files, you will also need to redistribute ISYSPDF6.DLL, ISYSPDFL.DLL, and ISYSPDFL.DAT, which should be placed alongside the ISYS8.DLL module.

For use with Lotus Notes natively, also redistribute the ISYSLN.DLL and ISYSLDS.DLL modules.

If your application makes use of Script EAMs, you should include the ISYSSCRIPT.DLL module.

If your application makes use of HTML templates, you should include ISYSHTML.DLL.

If your application indexes SWF files, you should include the ISYSWF.DLL module.

Your application should also include the ISYS.NLI and ISYS.CWD files, and also the ISYS.FLD, ISYS.SYN, ISYS.CAT, ISYS.SCT and ISYS.DEF files, if they exist.

If you are calling the ISYS engine via the COM interface, then you need to redistribute ISYSCOM.EXE and ISYSCOM.DLL, and you need to register both of these.

If you are using document content caching (via the CacheDocuments keyword in the ISYS.CFG) to accelerate document browsing and context displays at query time, then you need to redistribute ISYSCACHE.DLL and ISYSDC.DLL.

If you are calling the ISYS engine via Java, then you need to redistribute ISYSJAVA.DLL.

If you are using the ISYS Enforce\_Visible\_Security feature, you should also redistribute the ISYSAUTH.DLL module. If you are providing your own custom security filter, then distribute your own custom ISYSAUTH.DLL.

If you are building your own External Access Module (EAM), then you should either name your module ISYSEX32.DLL and place it in the executable directory, or name your EAM as you choose and explicitly load it in your ISYS.CFG file.

Please note that the ISYS8.LIC file in your ISYS:desktop directory is only relevant to ISYS:desktop, and should not be redistributed with your completed application.

It is important to review your ISYS.CFG before deploying it with your application, particularly if you have used ISYS:desktop to create your initial ISYS.CFG. ISYS:desktop enables a series of index options by default, notably number recognition, metadata caching, spelling tips and entity recognition, among others. Each of these options comes with a small performance cost in indexing speed, perhaps 10% per option. It is wasteful to have these options enabled if you are not externalizing the functionality in your application.

## 1.7 Indexing API

ISYS offers the choice of three different APIs for indexing information.

The simplest is to configure the ISYS engine in the same way you would using ISYS:desktop, setting configuration files such as ISYS.CFG, ISYS.SQL, ISYS.FTP and ISYS.MAL. Then you issue a single API call (IDB\_Function("UPDATE")), and ISYS automatically enumerates the document space, resolves differences, and brings itself up to date.

The most detailed method is via the Low Level Indexing API, where you must give the engine indexing directives on a file-by-file basis, very much programming “in the active voice”. This API involves numerous calls, careful programming, and strict conformance to protocols. In the vast majority of cases, the degree of control provided by the Low Level Indexing API is not required.

An intermediate indexing method is the Transactional Indexing API. This method is particularly good in situations where your application creates data asynchronously, but you wish the ISYS index to be updated more frequently than a periodic name-space scan would allow. You journal your changes to a transaction file, then apply the transaction file to the ISYS index with a single API call.

Most applications use IDB\_Function index updating; some use Transactional; very few find the need to use Low Level Indexing.

## 1.8 Retrieval API

Searching and retrieval is what ISYS is all about, and ISYS provides a very easy-to-use retrieval API that allows most developers to meet their objectives using less than a dozen API calls.

Chapter 3 shows the details of retrieving information with ISYS, but the general approach is to open an index, issue a query command, enumerate some or all of the result set, optionally open a document through the ISYS API, and optionally fetch hit positional information and document content for rendering.

## 1.9 WYSIWYG API

The WYSIWYG (“What You See Is What You Get”) feature defines a whole new subsystem within ISYS for the viewing of documents complete with full formatting. It operates according to a completely different mechanism, where your application initiates a viewing window, is given an hWnd to that window, and the window then operates semi-autonomously for viewing and user interaction purposes.

Depending on your usage, the WYSIWYG API may subject to separate licensing considerations and is documented in the chapter 17 of this manual.

## 1.10 Character Sets

ISYS normalizes text before it is added to the index. You may choose to normalize into ANSI or Unicode.

The documents ISYS reads may be represented in ANSI, Unicode, EBCDIC, OEM or MBCS, and ISYS will make appropriate conversions regardless of whether you have chosen to normalize your index as ANSI or Unicode.

Unicode is a good choice when your index is to contain multiple disparate languages, for example a single index containing Chinese, Russian and Spanish.

ANSI is a good choice when your index is to contain a single language, or a single language plus English. Put another way, an ANSI index is a good choice if your data can be represented in a single Windows codepage. For example, if your index is to contain Chinese and English, an ANSI index is sufficient.

Unicode indexes are more powerful and flexible, but carry additional overhead.

Applications may choose to interact with the ISYS API using either ANSI or Unicode. This decision is completely independent of the character set to which an index has been normalized.

See `Set_API_Charset` for information on how your application may interact with ISYS in ANSI or Unicode.

## 1.11 Dealing with messages in Visual Basic

ISYS uses Windows messages to pass back progress reports during searching, and also to stream message output from utility functions. For most languages, intercepting a message sent to a particular Windows handle is a simple matter. In Visual Basic, this may not be so.

In Visual Basic, sometimes the easiest way to intercept a progress message from ISYS is to make the message appear as changed text in a Visual Basic Textbox. By default, ISYS messages are sent with a message number of \$100, but you can change this to any value you like using `ISYS_Multiplex` with an `ISYS_Multi_SetMsg` parameter. Specifically, by changing the message value to `WM_SETTEXT`, and specifying the `hWnd` of a Textbox, the notification messages will just appear asynchronously in the Textbox and fire the Change event.

The Visual Basic sample program shows an example of this technique, which may also be used in other languages.

### Code Example:

```
' After the engine is initialized, change the notification
' message number to WM_SETTEXT (12)
Call ISYS_Multiplex(ISYS_Multi_SetMsg, 12, 0, ISYS_Error)

' perform the query. MyCallback is a textbox
Call Perform_Find("CAT", 0, MyCallback.hwnd, QRB, ISYS_Error)

...

Sub MyCallback.Change

' The ISYS message automatically appears in MyCallback.text
' Process the message as desired

Endsub
```

## 1.12 Using ISYS with non-English languages

ISYS works correctly with most single-byte non-English languages, including all major European languages.

By default, ISYS treats diacritical marks as significant from a searching point of view. In other words, characters with and without accents, or characters with accents, are seen as being quite distinct characters. You can optionally configure your ISYS index to consider diacritical marks to be not significant for the purposes of searching. If you do this, then accents are ignored in much the same way as casing.

ISYS also supports multi-byte Asian languages, including Korean, Traditional and Simplified Chinese, Hong Kong Chinese and Japanese. ISYS normalizes Asian characters to either MBCS or Unicode, depending on your index settings. Asian text in documents may be encoded in either MBCS (Big-5, Shift-JIS, GB, etc), or Unicode.

For Asian languages, ISYS applies appropriate understanding of word structure and delimiters. For example, ISYS can handle both ideographic and non-ideographic languages correctly.

You can mix multiple languages in documents and search expressions. For ANSI indexes, the combination of languages which may be used in a single index or single query is limited to what can be represented in a single codepage. For Unicode index, no such constraint exists.

## 1.13 Licensing and Support

Also included with your SDK kit is a licensing agreement. Please remember that you may make and distribute copies of the ISYS software only within the terms of the licensing agreement.

For SDK technical support, please contact ISYS Search Software. Addresses and telephone numbers are inside the front cover of this manual. Your normal ISYS support representative will put you in contact with the appropriate SDK support staff.



---

## 2. CONCEPTS

An ISYS *index* consists of up to 64 million indexed documents. A document normally corresponds to a word processor file. It may also be an ASCII file or correspond to a database record, or an email, or be some OEM-defined source of data that only your application knows about.

ISYS indexing can be done at two levels. At the highest level, a configuration file is created that contains a rule-base of how documents located on various volumes should be treated. The index is automatically brought up to date, with new documents indexed, and altered documents re-indexed. Call-backs advise of indexing progress. Only one indexing operation can be done at a time, but one indexing session and multiple queries may proceed at the same time. This functionality is described in chapters 3 and 4.

The second, lower level mechanism involves the host program using specific “index this file” API calls. The host program provides a “file name” of the file to be indexed. The file name need not be an actual filesystem filename, but can be considered a 255-byte access key that uniquely identifies the document. The host program is returned a 32-bit handle by which the index knows the document. The low-level indexing interface is described later.

There is also a transactional indexing mode which is ideal for applications which can “journal” their required updates to a transaction log which can be applied with whatever granularity is required, thereby offering the best trade-off between real-time updates and the efficiency introduced by granularity or batched operations.

An overview of the ISYS indexing process is as follows. All deleted and changed documents are identified and removed from the index en-masse. All new and changed documents are read, concorded, and added to the index. It is most efficient to perform index changes in bulk, rather than as they occur.

The retail version of ISYS:desktop is broken into two parts, the DLLs and a front-end. The ISYS DLL is the program that reads your documents, indexes them, and writes the index information into index files. The front-end is an application that communicates with the ISYS DLL and presents you with a user-friendly interface. Two front-ends ship with ISYS: ISYS Utilities and ISYS Query. The SDK allows you to write your own front-end. It comes with sample source code for a simple front-end written in Visual Basic, C, and Delphi.

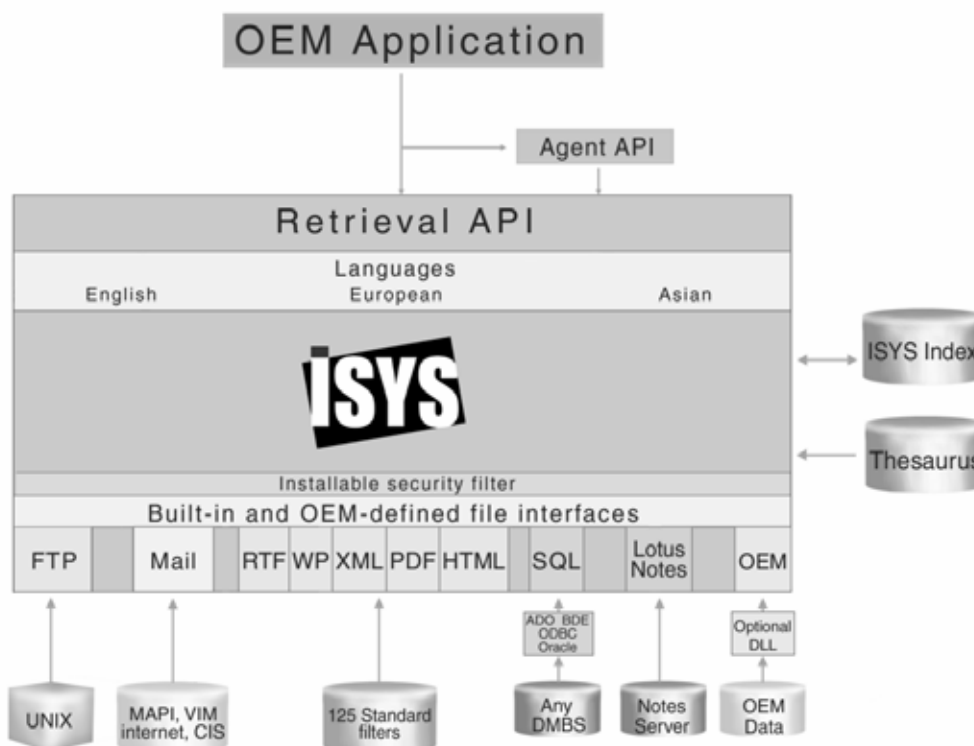


Figure 1. The program you write talks to the ISYS API. The DLL reads text files; it reads and writes the ISYS database.

## 2.1 Data Types and Parameter Passing

You communicate with ISYS by passing parameters through calls. This section describes the conventions used in this manual to describe how ISYS passes parameters. Even though you may be writing your front-end application in a different language from that in which the ISYS DLL was written, your application and the DLL can still communicate with one another as long as the data types they use and expect are properly defined. The ISYS DLL was written in Delphi, so the data types in this manual will follow Pascal naming conventions. Here are the fundamental Pascal data types, their ranges, and their binary representations:

<b>Data Type</b>	<b>Range</b>	<b>Format</b>
Byte	0..255	Unsigned 8-bit (1 byte)
Shortint	-128..127	Signed 8-bit (1 byte)
Word	0..65535	Unsigned 16-bit (2 bytes)
Smallint	-32768..32767	Signed 16-bit (2 bytes)

Note: Throughout this manual, we try to avoid using the term “Integer” which can sometimes be ambiguous. Instead, we explicitly refer to either “Longint” (32-bit) or “Smallint” (16-bit).

<b>Data Type</b>	<b>Range</b>	<b>Format</b>
Longint	-2147483648..2147483647	Signed 32-bit (4 bytes)
Pointer	N/A	Unsigned 32 bits of address data (4 bytes)
Handle	N/A	(32-bit, 4-byte token)

*Strings* (groups of printable characters) are represented in one of three ways:

- *ASCIIz* strings start with a pointer to the first character and are terminated by a single byte of value 0. Their size is limited only by the amount of memory in your computer, however, they cannot contain zero bytes as part of the string. The C programming language commonly uses ASCIIz strings.
- *Fixed-length* strings contain a fixed number of characters. Routines exchanging fixed-length strings must agree through some other mechanism how long strings will be.
- In a few cases, *Pascal* strings are used which start with a byte that tells how many characters follow; they are limited to 255 characters.

Note: depending on how you have chosen to interact with the ISYS API, strings may contain either ANSI or Unicode characters. In the case of Asian multibyte languages, the ANSI characters will be interpreted as MBCS where appropriate. In the case of Unicode, the encoding is UTF-8, wherein each Unicode character occupies between one and six bytes. In either case, for ASCIIz strings a single zero terminates the string regardless of whether it is ANSI, MBCS or Unicode. Thus strings can be processed normally and it is not necessary to know the character size in order for you to handle the string correctly.

Following the Pascal convention, a subroutine that returns a value through the name of the routine is called a *Function*. A subroutine that performs an action and does not return a value is called a *Procedure*. Different programming environments have different names for these concepts:

<b>Pascal</b>	<b>C</b>	<b>Visual Basic</b>
Function	function()	Function
Procedure	(void) function()	Sub

Included in the SDK are files containing data types and subroutine definitions in C, Delphi, and Visual Basic. Include the appropriate files with your source code: all the definitions will exist as they are described in this manual.

Parameters to functions and procedures can be given in two ways: by value, where the

contents of a parameter is given to the subroutine; or by reference, where the *address* of the parameter is given to the subroutine. For a small parameter such as an integer, it is faster to pass it by value. Changes made to the parameter in the subroutine are not seen outside the subroutine; the original copy of the parameter retains its old value. For a large parameter such as a string or a function result block, it is faster and takes less memory to pass it by reference. Because the function or procedure has the address of the parameter, it can modify the parameter.

## Passing Parameters

Different programming environments express parameters in different ways

<u>Delphi</u>	<u>C</u>	<u>Visual Basic</u>
MyInt: smallint	int	ByVal MyInt%
MyWord: wordint	WORD	ByVal MyWord%
MyLong: longint	long	ByVal MyLong&
MyString: pstr	LPSTR	ByVal MyString\$
var MyInt: smallint	LPINT	MyInt%
var MyLong: longint	LPLONG	MyLong&
var MyStruct: structure_type	structure_type FAR*	MyStruct as structure_type

If you are using a programming language not listed here, you will have to become familiar with its calling conventions and determine how they map to the conventions above.

## 2.2 Result Blocks

Often parameters are passed back and forth in data objects called records, structures, or blocks. Several calls to the ISYS API use blocks to return data to your application. ISYS control blocks are defined with the API calls that first use them.

The great majority of calls in the ISYS API report errors in an error message control block. An error message control block consists of a single ASCIIz string up to 238 characters long., followed by a two byte error code which classifies the error. Your program must create a structure for these result blocks and pass it as a parameter to every API routine you call. If the call worked, the first character is 0. If the call didn't work, the string contains an error message. Note that ISYS cannot detect what kinds of parameters you are passing. Giving ISYS the wrong kind of parameters is generally fatal to the program.

**Error\_Control\_Block**

Control block used to return error messages from ISYS.

Msg            ASCIIz: error message string, up to 238 bytes long

MsgID        Word: error message code

## 2.3 The Sample Applications

Included in the SDK kit are a collection sample applications for each of C++, C#, Java, VB, VB.Net, VBscript and Delphi.

These files are a basic example of how to write a custom front-end for ISYS. They are very limited in what they can do, but show you how to call the ISYS Engine without a lot of confusing detail.

While the sample applications are not intended as a platform for further development, they are an excellent way for you to learn the basic concepts of the ISYS Engine.

Most of these samples are quite minimalist and just illustrate the basic calling sequence to open an index, perform a query and list the results.

Note that all applications must pass ISYS a License Code. The sample programs include template code to pass an License Code to the ISYS engine, but the License Code is not actually provided as part of the program code. You must insert your provided License Code into the sample programs. If you have not done so, you will receive a compilation error.

## 2.4 License Codes

All applications calling the ISYS engine must quote a License Code in their first call to `Init_Instance` (described in the following section).

If you are evaluating the ISYS SDK, a License Code will have been provided to you which you may use with the sample programs. This License code will be time-limited, and may also cause the ISYS engine to display a message upon usage. However, this License Code may be used from any executable name.

If you have purchased the SDK and plan to ship your own application, you will have been provided with one or more permanent License Codes. These codes are tied to your executable name, and will only be valid when called from the associated executable name. They are not time-limited when used within your compiled application, but may be time limited when used within an IDE or with a debugger attached, depending on your license agreement with ISYS Search Software.

If you are developing in an interpretive language such as Visual Basic, you may find that your executable name when running inside the development IDE is not your program name. For example, a Visual Basic project which generates an executable named MYAPP.EXE only runs with this name when running outside the IDE. When run within the IDE, the application name will be something like VB.EXE. In these situations, either run your application outside the IDE, or if you need to run inside the IDE for debugging purposes, use an evaluation Authorization code which will run against any EXE. Fully compiled languages usually do not have this issue.

---

## 3. BASIC RETRIEVAL

The routines described in this chapter are the minimum required to build a basic query tool. Here is a list in approximately the order that your program will call them.

<u>API Routine</u>	<u>Description</u>
Init_Instance	Opens a session with ISYS.
Get_Database_Name	Gets information about an ISYS database.
Open_Database	Opens an ISYS database.
Perform_Find	Submits a query to ISYS.
Perform_English_Find	Submits a Plain-English query to ISYS.
Sort_Documents	Sorts a list of files that meet the query criteria.
Get_Document_Record	Get information about a document.
Open_Document	Prepares a document for examination.
Get_Line_With_Hit_Before	Goes to the previous query-hit in the document.
Get_Line_With_Hit_After	Goes to the next query-hit in the document.
Get_Document_Line	Gets a specified line from a document.
Close_Document	Finishes a document examination.
Close_Instance	Closes a session with ISYS.

For each description, the first item is the name of the routine. This is followed by a description and the purpose of the routine, what it does, and its parameters. If one of the parameters is a special data structure, that structure is defined in a box after the individual parameters. Following some descriptions is a code example that shows how to call the function or procedure from Visual Basic.

### 3.1 Procedure Init\_Instance

The first call to the DLL by any application must always be Init\_Instance.

<u>Parameters</u>	<u>Description</u>
hWnd	Handle to a window that will receive messages from the ISYS Engine, or zero if no such messages are required.

Estr	Pointer to a zero-terminated (ASCIIz) string that contains the “base” or “executable” directory. Whenever ISYS searches for configuration files, it will include this directory in its search. This is usually the directory of the calling application and the directory <b>must</b> contain a valid ISYS7.KEY file.
InstBlk	Pointer to an Instance_Status_Block. This block must be filled with your application License Code <i>prior</i> to the call, and will return with your licensee information <i>after</i> the call.
InstHandle	<p>A pointer to a 16-bit smallint which will be filled with an instance handle. Where you need to instantiate multiple instances of the ISYS Engine from the one instance of an application program, this handle can be used to alternate between instances. See Switch_Instance for more information.</p> <p>If you do not need multiple Engine instances per application instance, you do not need to make use of this parameter again, and can just ignore the value returned.</p>
EmsgBlk	Pointer to Error_Control_Block.

### Instance\_Status\_Block

Control block returned by Init\_Instance with information about the installed ISYS Engine:

DLL_Version	16 byte fixed-length string: contains DLL version number
Reserved	Word: Reserved
License_Type	Word: indicates the license type in use
Licensee_ID1	40 byte fixed-length string: the user name you entered when you installed ISYS. Before Init_Instance is called, this field must be loaded with your License Code.
Licensee_ID2	40 byte fixed-length string: the company name you entered when you installed ISYS

Before the call to Init\_Instance, copy your License Code into the Licensee\_ID1 field of the Instance\_Status\_Block. If your License code is invalid or expired, the call to Init\_Instance will fail and an error will be returned in the Error\_Control\_Block. If the code is valid, your Instance\_Status\_Block will be filled accordingly.



The window specified by the `hWnd` parameter is only sent messages that communicate progress information and messages from `IDB_Function`. (The messages returned by `Perform_Find` are sent to the window specified in that call.) Progressive results from queries (not database utility functions) are passed back to the window specified in the `hWnd` parameter. In C and Pascal, use standard Windows techniques to retrieve the message. If you are programming in Visual Basic, see the section “Dealing with messages in Visual Basic”.

The `Init_Instance` call returns a handle to the instance, and multiple instantiation per program is quite valid. Subsequent API calls talk to the active instance, and you use the `Switch_Instance` API call to direct subsequent calls to the various instances you may have active. You only need to call `Switch_Instance` when you want to change which instance you are talking to. It is still your responsibility to close all opened instances in an orderly fashion. Note this is only an issue where one instance of one program plans to make multiple `Init_Instance` calls without closing the intervening instances.

**Code Example:**

```
' Initialize the ISYS Engine.
MyInstanceBlock.Licensee_ID1 = "HAGTQ KJSHD AIUAA KSJHA DHGGD"
Call Init_Instance(0, "C:\myprog", MyInstanceBlock, ISYSError)
' 0 means that VB is not expecting messages from ISYS.
' C:\myprog is where ISYS is installed on this machine.
' This is where the DLL will look for the ISYS.KEY file.
' MyInstanceBlock is declared in Globals.
' ISYSError is declared in Globals.
```

## 3.2 Procedure Open\_Database

Once the ISYS Engine has been initialized, you tell it in which index to perform searches by using the Open\_Database call. The Update\_Mode parameter specifies whether the index is to be opened in read-only or read-write mode. If the index cannot be opened, the reason is given in the error message control block. If synonym, concept, or saved-query lists exist, they are loaded. If another index is open, that index is automatically closed first.

<u>Parameters</u>	<u>Description</u>									
DirStr	Pointer to ASCIIz directory string: contains a directory or a list of directories delimited by semicolons. A maximum of 128 indexes may be opened at once in a single Open_Database call, unless you have opted to increase this limit from its default.									
Update_Mode	SmallInt: value that determines whether the index is opened read-only or read-write: <table><tr><th><u>Constant</u></th><th><u>Value</u></th><th><u>Definition</u></th></tr><tr><td>ISYS_OpenReadOnly</td><td>0</td><td>ISYS won't change index files.</td></tr><tr><td>ISYS_OpenReadWrite</td><td>1</td><td>ISYS may write to the index files.</td></tr></table>	<u>Constant</u>	<u>Value</u>	<u>Definition</u>	ISYS_OpenReadOnly	0	ISYS won't change index files.	ISYS_OpenReadWrite	1	ISYS may write to the index files.
<u>Constant</u>	<u>Value</u>	<u>Definition</u>								
ISYS_OpenReadOnly	0	ISYS won't change index files.								
ISYS_OpenReadWrite	1	ISYS may write to the index files.								
EmsgBlk	Pointer to Error Control Block									

### Code Example:

```
Call Open_Database(IndexFilename, ISYS_OpenReadOnly, ISYSError)
```

## 3.3 Procedure Get\_Database\_Name

ISYS indexes can be given descriptive names. This routine extracts the full index name for the index you specify. Given a pointer to an ASCIIz directory name and a pointer to a buffer, Get\_Database\_Name fills the buffer with one of the following:

- an ASCIIz string index name as extracted from the ISYS.CFG file in that directory,
- a constructed name if no name was provided in the ISYS.CFG file, or
- an appropriate message if no CFG file exists in that directory.

If you pass a list of directory names delimited by semicolons, you will be returned a list of index names also delimited by semicolons. The name of each entry will be determined as though it were the only entry and it had been passed by itself rather than in a list.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
DirStr	Pointer to ASCIIz directory string: contains a directory or a list of directories delimited by semicolons
NameStr	Pointer to buffer of at least 256 bytes: filled with the index name from the specified directory. You must allocate the 256-byte buffer in your program.
EmsgBlk	Pointer to Error_Control_Block

**Code Example:**

```

IndexName = String$(255, 0)

Call Get_Database_Name(IndexFilename, IndexName, ISYSError)

' Display that information on the main form.

LabelIndexName = "Using Index " & IndexName & " in " &
: IndexFilename

```

### 3.4 Procedure Perform\_Find

This is the central call in the API: everything else exists to make this call possible. Call Perform\_Find to have ISYS execute a query. Perform\_Find takes a query string and fills the Query Result Block with the statistical results of the search. As the query proceeds, it passes messages to the window specified by hWnd. If a result-set currently exists from a previous query, that set is discarded. In other words, a result-set is only valid until another query is performed. The language for specifying queries is described in the ISYS Query help.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
QueryStr	Pointer to an ASCIIz string: contains a standard ISYS query to be executed. This string may be of arbitrary length, but of no more than 1024 lexical tokens per query.
wFlags	Word: quantity that specifies how the query is to be processed. Use the following constants defined in the header files:

<b>Constant</b>	<b>Value</b>	<b>Definition</b>
ISYS_Query_Synonym	1	synonym expansion
ISYS_Query_Conflate	2	verb conflation
ISYS_Query_Thesaurus	8	thesaurus expansion
ISYS_Query_Internet_Syntax	16	alternate query syntax
ISYS_Query_AND_With_Current	32	automatic AND with current result
ISYS_Query_Synonym+ISYS_Query_Conflate	both	
ISYS_Query_None	0	none
hWnd	Handle of the window that is to receive progress notification messages as the query proceeds. If no such notifications are required, this can be zero.	
QRB	Pointer to a caller-allocated Query_Result_Block filled by the DLL	
EmsgBlk	Pointer to Error_Control_Block	

### Query\_Result\_Block

Control block filled by Perform\_Find. It contains the results of the search.

Total_Hits	Longint: number of hits
Diff_Words	Longint: number of different words
Num_Docs	Longint: number of documents
Filtered_Flag	Boolean word: true if result has been filtered
Subquery_Size	Longint: if the query is a sub-query, this value is the size of the query-set that this query was performed within
Phrased_Hits	Longint: number of hits, with hits on consecutive words counted as a single hit. This can give a more intuitive hit count where phrases are involved.

**Code Example:**

```
wflags = ISYS_Query_Synonym + ISYS_Query_Conflate

Call Perform_Find(TextBoxQueryString, wflags,

    FormEnterQuery.Callback.hWnd, MyQueryResult, ISYS_Error)
```

The Windows message, by default, is sent as number \$100, although you can change this via an ISYS\_Multiplex call. The wParam message parameter set to a handle to a global memory block that contains an ASCIIz string, and the lParam is set to a pointer to that same locked global memory block, and can be directly referenced. The string is formatted as follows:

```
"<msg> <word> <tab> <occs> <zero>"
```

Where <msg> is a single character: "H" indicates that the search is proceeding; "B" indicates that a large Rlist is being calculated. <word> indicates the word portion of the query most recently completed, <tab> is a delimiter, and <occs> is a string representation of how many occurrences of the <word> were retrieved. These messages are generated in real-time as the query is evaluated.

Refer to the Windows API documentation for information about Windows messages. The means by which you deal with Windows messages will depend upon your development language. You may choose to pass an hWnd of zero and ignore these messages entirely, however you will then not receive feedback as the query evaluation proceeds, and will be unable to call Interrupt\_Processing if required.

### 3.5 Procedure Perform\_English\_Find

Performs a Natural Language (Plain English) Query. Like Perform\_Find, it takes a query string and fills the Query Result Block with the statistical results of the search. As the query proceeds, it passes messages to the window specified by hWnd. If a result-set currently exists from a previous query, that set is discarded. In other words, a result-set is only valid until another query is performed. An important difference from Perform\_Find is that Perform\_English\_Find does not have the wFlags parameter.

Natural Language query results are always automatically sorted by relevance.

<b>Parameters</b>	<b>Description</b>
QueryStr	Pointer to an ASCIIz string: contains a natural language ISYS query to be executed. The maximum length of a natural language query string is 255 characters.
hWnd	Handle of the window that is to receive progress notification messages as the query proceeds. If no such notifications are required, this can be zero.  See the section on Perform_Find for details on how to retrieve and interpret the messages.
QRB	Pointer to a caller-allocated query result block: filled by the DLL
EmsgBlk	Pointer to Error_Control_Block.

**Code Example:**

```
Call Perform_Find(TextBoxQueryString,
    FormEnterQuery.Callback(hWnd, MyQueryResult, ISYSError)
```

Natural Language lets people type in exactly what they need to know in plain, everyday English, without the complication of trying to formulate a query command or make the conceptual translation into operators and search terms. Please consult the ISYS:desktop help for examples of Natural Language and how it is used.

## 3.6 Procedure Sort\_Documents

You can display the list of documents that satisfy the search requirements in several different orders. Sort\_Documents is the routine you call to make this happen.

Sort\_Documents sorts the current result-set by a criterion you specify. Documents in the result-set are still numbered 1 to QRB.Num\_Docs, but the document assigned to each ordinal will change. Sorting may require the ISYS Engine to internally determine the Document Control Block for each document in the result-set, and so may take some time additional time. If the result-set is not sorted, it is presented in the order the files were added to the index.

The default order in which files are listed is 1, the indexed order. No actual sorting takes place for this order. To sort the document list to any other order, the ISYS Engine may have to look up each file in its indexes to get the information the sort is based on. These lookups may add to the time it takes to finish the sort, depending on which sort method you use.

**Parameters Description**

SortSeq      Smallint: specifies the sort sequence. Use the following constants defined in the header files:

<b><u>Constant</u></b>	<b><u>Value</u></b>	<b><u>Definition</u></b>
ISYS_Sort_Default	1	Default database (file system) order.
ISYS_Sort_NumHits	2	Number of hits in document. Hits on multi-word phrases are counted as one.
ISYS_Sort_DocSize	3	Size of document in words.
ISYS_Sort_FilePath	4	Full file path.
ISYS_Sort_FileType	5	File extension.
ISYS_Sort_FileName	6	File Name.
ISYS_Sort_DateTime	7	Date/Time stamp.
ISYS_Sort_Relevance	8	Relevance quotient.
ISYS_Sort_DocDate	9	First intelligent date appearing in document.
ISYS_Sort_Bytes	10	Size of document in bytes.
ISYS_Sort_Indexed	11	Date indexed.
ISYS_Sort_Title	12	Document title.
ISYS_Sort_UnPhrasedHits	13	Number of hits in document, but hits on multi-word phrases are individually counted.
ISYS_Sort_Format	14	Document format.
ISYS_Sort_Metadata	15	See Set_Metadata_Sort_Key
ISYS_Sort_Precedence	16	Position of the earliest hit within the document.

ISYS_Sort_Cat_Weighted_Relv	17	Relevance quotient, weighted by the category relevance.
ISYS_Sort_Within_Index	32	Hierarchical sort; outer level is by chained index.
ISYS_Sort_Within_Category_Name	64	Hierarchical sort; outer level is by category name
ISYS_Sort_Within_Category_Freq	128	Hierarchical sort; outer level is by most populous category.
ISYS_Sort_Within_Category_Relevance	192	Hierarchical sort; outer level is by most relevant category
ISYS_Sort_Reverse	-1	Multiply another sort sequence by this constant to sort in reverse sequence.

EmsgBlk     Pointer to Error\_Control\_Block.

**Code Example:**

```
Call Sort_Documents(ISYS_Sort_NumHits, ISYSError)
```

### 3.7 Procedure Get\_Document\_Record

After a query has been performed and the QRB has been set to indicate that QRB.Num\_Docs documents have been retrieved, this call may be used to retrieve details about a particular document in the retrieval set. Documents in the set are numbered from 1 to QRB.Num\_Docs. ISYS does not determine details about the found documents until you specifically request them. Document records may be accessed in any order.

<b>Parameters</b>	<b>Description</b>
DocNum	Longint: in the range 1 to QRB.Num_Docs
DocBlk	Pointer to a caller-supplied Document_Control_Block: will be filled by the DLL. The DCB is formatted as follows.
EmsgBlk	Pointer to Error_Control_Block



**Document\_Control\_Block**

The Document Control Block is filled by a call to Get\_Document\_Record.

Document_Num	Longint: index document id. This is passed back for information only. Your application will probably not use this. It represents the unique document number within the index. In the case of a chain of indexes, this is a synthetic number arranged to be temporarily unique throughout the chain.
File_Path	ASCIIz: document name (255 bytes). May be an operating system file name, URL, FTP URL, email message identifier, XML file name and record position, SQL key field, or arbitrary OEM document identifier. The file name is returned in a form suitable for visual use by the user. Absolute file names are returned in full, while for documents indexed as relative, only the relative portion is returned. If the document name includes any internal information, this is stripped.
Document_Title	ASCIIz: ISYS document title (150 bytes). This defaults to the first text line of the document, or the formal metadata title, but ISYS can be configured to select lines that contain specific strings.
Document_Words	Longint: number of words in document.
Num_Hits	Longint: number of hits in document.
Phrased_Hits	Longint: number of hits, with hits on consecutive words in a phrase counted as a single hit. This can give a more intuitive hit count where phrases are involved.
Relevance	Longint: an expression of the calculated relevance of this document compared to the other documents in the same result set. Ranges from zero to 100, where 100 is the most relevant. The calculation is based on hit clustering, cluster proximity, document density, word weight, metadata and other factors. Relevance is a comparative measure amongst the documents in the same result set.
File_Format	Longint: document format code, as listed in Appendix A.
File_Time_Stamp	Longint: document file date/time stamp or generation counter.
Size_in_Bytes	Longint: document file size in bytes.
When_Indexed	Longint: date/time stamp of when the document was indexed.
When_Confirmed	Longint: date/time the existence of the document was last confirmed (applies to spidered websites only).

Date_in_Document	Longint: the first date appearing in the text of the document.
HTML_File_Format	Longint: in the case of documents which have been dynamically converted to HTML, this shows the true format of the underlying file, as listed in Appendix A.
Chain_Member	Longint: in the case of a chained index, this indicates in which member of the chain the document was found.
SubChain_Document_Num	Longint: in the case of a chained index, this indicates the true (unsynthesised) Document_Num, which is unique within the index, but not necessarily within the chain.
NonUnicode_Codepage	Longint: in the case of non-Unicode documents processed into a Unicode index, this indicates the codepage which was used for this document. The codepage used may have been one explicitly stipulated by the document itself (for example, a CHARSET directive in an HTML page), or one specified to be assumed via the ISYS.CFG, or failing that, the default codepage of the indexing machine
Full_File_Name	ASCIIz: full document name (255 bytes). Fully qualified (even if relative), and including any internal information that would not normally be shown to the user for visual purposes.
Category	ASCIIz: the category to which the document has been assigned.
Metadata_Hits	Longint: the number of hits which occurred within the metadata portion of the document.
Metadata_Lines	Longint: the number of lines of metadata present at the top of the document.
Paragraph_Count	Longint: the number of paragraphs in the document.
Flags	Longint: flags Bit 1 set if document is an email and an attachment exists Bit 2 set if document is annotated
Entity_Count	Longint: number of entities detected

**Code Example:**

```
'Get information about all the hits in the list
'and put it in the list box on screen.
For index = 1 To MyQueryResult.Num_Docs
    Call Get_Document_Record(index, MyDocBlk, ISYSError)
    Call HandleError("Globals.ShowTheHitList")
    FormEnterQuery.ListHits.AddItem
        pad(Str$(MyDocBlk.Words_in_Document), 5) &
        pad(Str$(MyDocBlk.Number_of_Hits), 5) & " " &
        RTrim(MyDocBlk.File_Path) & "; " &
        RTrim(MyDocBlk.File_Title)
Next
```

### 3.8 Function Open\_Document

Once you have selected a document from the hit-list, call Open\_Document to open a document in the result-set and get a reference to it. Since you may open more than one document at a time, every call should quote the reference that Open\_Document returned.

<b>Parameters</b>	<b>Description</b>
DocNum	Longint: the document number in the range 1 to QRB.Num_Docs
EvenIfChanged	Smallint: when nonzero, forces the document to be opened even if the document has changed since it was indexed. Documents that have changed since indexing cannot have their hits highlighted.
NumLines	Pointer to a Longint. For non-WYSIWYG documents, will be set to reflect the number of lines in the document. Lines within the document may then be accessed by ordinal 1 to NumLines. For WYSIWYG documents, this will be set to a negative number that represents the controlling hWnd of the WYSIWYG viewer. See the chapter on WYSIWYG viewing for more information.
FmtCode	Pointer to a Smallint: to be set to the ISYS document format code. See Appendix A: Document Format Codes.
FnameStr	Pointer to a 255-character ASCIIz buffer. Open_Document fills this with the fully qualified physical file name of the document. In the case of a ZIPped document, or a document from an HTTP Internet source, this may be a temporary file name.
EmsgBlk	Pointer to Error_Control_Block
<b>Returns</b>	Smallint: a unique identifier for that document

<b>Constant</b>	<b>Value</b>	<b>Definition</b>
ISYS_Open_OK	0	If the value returned <i>is greater than</i> this constant, the document was opened and the value is the document handle.
ISYS_Open_Failed	-1	The document could not be opened.
ISYS_Open_Changed	-2	The document has changed since indexing.
ISYS_Open_NoRights	-3	The document could not be opened because it is already open with exclusive access or you do not have sufficient rights to open the document.

ISYS_Open_NoReader -4	The document is of indeterminate format (a “late bound Auto”), and at the time of the open, was not found to be of a format ISYS is able to read.
ISYS_Open_Password -5	The document is password protected and cannot be opened.
ISYS_Open_CacheError -6	Content caching is enabled and there was an error accessing the cache.

If `Open_Document` returns `ISYS_Open_Changed`, you may issue the open again with `EvenIfChanged` set to a non-zero value. If it returns a positive number, it is the document handle. You give this value back to ISYS routines in the `hDoc` parameter.

While the value returned by this routine is a 16-bit integer, the parameter required by the other routines is a 32-bit longint. This discrepancy has historical reasons. Since the values returned by this routine are generally less than or equal to 48, the different integer formats will not cause a problem. Simply assign the result of this function call to a longint variable and pass that variable back in subsequent calls.

**Code Example:**

```
dim idocnum as integer

dim Ldocnum as long

dochandle = Open_Document(idocnum, EvenIfChgd, numlines,
fmtcode, linestr, ISYSError)

Ldocnum = idocnum
```

### 3.9 Function Get\_Line\_With\_Hit\_After

### 3.10 Function Get\_Line\_With\_Hit\_Before

When a non-WYSIWYG document has been opened, you find out where the hits are with these two routines. Given a line number, these routines return the number of the next or previous line in the document that contains a hit. To find the first hit in a document, call `Get_Line_With_Hit_After` with `LineNo = 0`. To find the last hit in a document, call Function `Get_Line_With_Hit_Before` with `LineNo = Num_Lines+1`.

These routines navigate from hit to hit, without distinction between what search term the hit represents. For example, if your query was “dog or cat”, these routines will locate hits regardless of whether it is a cat or a dog. If you wish to do term-specific navigation, use `Get_Line_With_Term_After` or `Get_Line_With_Term_Before` instead.

Both routines have the same parameters:

<b><u>Parameters</u></b>	<b><u>Description</u></b>
hDoc	Longint: document reference
LineNo	Longint: line number
EmsgBlk	Pointer to Error_Control_Block
<b>Returns</b>	Longint: line number or zero if no such hit exists

It is important to make sure that LineNo is within the bounds

$0 < \text{LineNo} \leq \text{NumLines} + 1$ .

hDoc must be a valid document reference returned by Open\_Document.

**Code Example:**

```
If dochandle > 0 And linenum < numlines Then
    linenum = Get_Line_With_Hit_After(dochandle, linenum,
    ISYSError)
    If linenum = 0 Then
        linenum = numlines
    End If
    Call show_document_line
Else
    Beep
End If
```

### 3.11 Procedure Get\_Document\_Line

Non-WYSIWYG documents are accessed in a line-centric fashion.

When you know the number of a line in the document you wish to display, call this routine to extract the line from the file. You can think of an opened document as an array of lines. Since the call to Open\_Document tells you the number of lines, you can access the lines in any order you want.

Given a reference to an opened document, Get\_Document\_Line returns a line from the document. As a way to retrieve data very quickly, ISYS artificially formats documents into lines and pages while indexing. This formatting is compatible between all platform versions of ISYS. By default, ISYS will wrap lines at 76 characters if they do not have soft returns. You can declare a larger notional right margin by use of the WIDELINES indexing option. Using WIDELINES on its own enforces a 200 character wrap. For an arbitrary margin, use the WIDEMEANS option. Use of these options is described in the ISYS Utilities help.

You may access document line numbers in any order; the Engine maintains its own internal cache and can navigate randomly with great efficiency. In other words, you may access line 400,000 in a 500,000 line document, and ISYS will not have to read from the top of the document in order to reach the required line.

<u>Parameters</u>	<u>Description</u>
hDoc	Longint: reference returned when the document was opened
LineNo	Longint: within the range 1 to NumLines (returned by Open_Document)
LineStr	Pointer to a 256-byte buffer: will be filled with an ASCIIz string representation of the line. The line may contain special marker characters as described below.
EmsgBlk	Pointer to Error_Control_Block

Within LineStr, special codes are used to indicate four font styles: underline, *italic*, **bold**, and “hit.” These style codes let you show the text with the same text styles it had in the original document. Two styles cannot coexist, however; when a new style is specified, any previous style is turned off.

Note that line length is specified in characters, not bytes. For Unicode indexes, UTF8 characters may occupy more than one byte. Thus even when the line length is 76 characters, for a Unicode index, more than 76 bytes may be returned in the line. For this reason, always pass LineStr pointing to a 256-byte buffer, even if you can be sure your line length is smaller. ISYS will never put more than 255 bytes in a line.

<u>Style Code</u>	<u>Value</u>	<u>Definition</u>
ISYS_Char_SoftSpace	1	Soft space
ISYS_Char_Normal	2	Start normal style
ISYS_Char_Bold	3	Start <b>bold</b> style
ISYS_Char_Italic	4	Start <i>italic</i> style
ISYS_Char_Underline	5	Start underline style
ISYS_Char_Hit	6	Start “hit” style
ISYS_Char_Paragraph	16	New paragraph marker
ISYS_Char_Start_Entity	17	Entity starts
ISYS_Char_End_Entity	18	Entity ends

A soft space is used to pad words when fully justifying text with a monospaced font, or when expanding tabs.

A special case is character code 6, which indicates the start of a “hit”. The character code 6 is immediately followed by another character which represents the term id of the word found. Term ids are numbered starting from 48, which for convenience is the code for the visible character “0”.

For example, if you searched for “cat or dog”, and if appropriate synonyms were in effect, you might see contents such as:

```
#6 #48 CAT #2 .... #6 #49 DOG #2 .... #6 #48 FELINE #2
```

Note that every hit begins with a character code 6 and ends with a character code 2. Following each 6 is the term id, which is 48 for “cat” and 49 for dog. The Get\_Term\_Id\_Term routine may be used to map term id’s to actual search term words used in the query. Also note that this query has found a hit on “feline” (due to a synonym, for example), and that this is identified as a hit for term id 48. In other words, “cat” and “feline” both have the same term id because even though they are different words, they have been found because of the same search term.

Another special case is entity recognition, if enabled. When an entity appears in the document stream, it is prefixed by a character code 17, which is immediately followed by a four character entity ID. Then the normal text of the entity follows as it would normally appear. Finally, the extent of the entity is terminated by a character code 18. For example:

```
THE SENDER WAS #17 AAAA MR SMITH #18 ON TUESDAY
```

In this example, the entity ID is the four characters ‘AAAA’, which immediately follows the #17 marker. Get\_Document\_Entity\_Record may be called to obtain the details of an entity



from its ID, and `Get_Line_With_Entity_After` may be used to navigate to the next occurrence.

It is the calling applications responsibility to render the found lines onto the screen. Typically this is done using a standard Windows scrollbar whose range is dimensioned from 1 to the number of lines in the document. The paint handler for the Window then simply calls `Get_Document_Line` in a loop from 1 to the number of lines on the screen (plus the current scrollbar value), and renders each line accordingly. The simplest way of doing so without displaying any attributes is to remove all characters less than or equal to 16 from the string (and to remove the character immediately following character code 6), remove code 17 and the four characters which follow, and remove character 18. Then use a simple `Textout()` to display the string. More sophisticated applications will want to change the text attributes to reflect hits.

There is a good example of the code to render a line in the Delphi sample program.

**Code Example:**

```
'Get the line before the hit.

If linenum > 1 Then

    linestr = String$(256, 0)

    Call Get_Document_Line(dochandle, linenum - 1, linestr,
    ISYSError)

LabelFoundText = StripStyleCodes(linestr) + " "

End If

'Get the line with the hit.

linestr = String$(256, 0)

Call Get_Document_Line(dochandle, linenum, linestr, ISYSError)

LabelFoundText = LabelFoundText + StripStyleCodes(linestr)


'Get the line after the hit.

If linenum < numlines Then

    linestr = String$(256, 0)

    Call Get_Document_Line(dochandle, linenum + 1, linestr,
    ISYSError)

LabelFoundText = LabelFoundText + " " +
StripStyleCodes(linestr)

End If
```

### 3.12 Procedure Close\_Document

When you are finished with a document, call this routine. It closes a previously opened document, releases associated memory, discards annotations, and closes the document file. If a temporary file was created from a ZIPped document, it erases the temporary file.

<u>Parameters</u>	<u>Description</u>
hDoc	Longint: document reference number
EmsgBlk	Pointer to Error_Control_Block

#### Code Example:

```
Call Close_Document(dochandle, ISYSError)
```

### 3.13 Procedure Close\_Instance

When you are finished calling the ISYS Engine, call this routine. It advises the DLL that you are finished with its services. The Engine will release all memory associated with that client and terminate any outstanding processes. All applications that open communications with the ISYS DLL must make this call, otherwise you may eventually run out of memory.

<u>Parameters</u>	<u>Description</u>
EmsgBlk	Pointer to Error_Control_Block

#### Code Example:

```
Call Close_Instance(ISYSError)
```



---

## 4. ADVANCED RETRIEVAL

The routines described in this section extend the capabilities of ISYS beyond the basic query tool. The calls are listed approximately in the order in which you would call them.

<b><u>API Routine</u></b>	<b><u>Description</u></b>
ExistOnPath	Gets the path to a file
FileName_Filter	Filters queries based on file name or date
Set_Sub_Find	Restricts subsequent searches to found documents
Set_Rlist_Curfew	Limits the number of hits that ISYS will report
Word_Search	Searches for a word in the index
Remove_Document_Record	Removes a document entry from the current list
Interrupt_Processing	Halts a search in progress
Get_Hit_Pointer	Gives detailed information about the query
Get_Document_Type	Tells what kind of document it is
Get_Document_Fully_Qualified_FileName	Returns the complete document file name
Get_Document_Long_Filename	Returns the long form of the filename
Get_Format_Attribute	Converts document type code to text
Get_Document_Time_Stamp	Tells when the document was last modified
Get_Document_Line_Page_No	Returns the page number for a given line
Get_EAM_Context_Info_for_Hit	Returns character position for a given hit
Auto_Determine_File_Format	Determines the format of a file
Translate_Hyperstring	Creates an ISYS query string
Open_Unfound_Document	Opens an arbitrary document
Close_Find	Closes a query and discards the result-set
Close_Database	Closes the current database and clears result-sets

## 4.1 Procedure ExistOnPath

Given a file name, ExistOnPath searches in order of the current directory, the application directory, and all the directories on the operating system path. If it finds the file in any of these places, it returns its complete path and file name. Otherwise, it returns a null string.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
FileNameStr	Pointer ASCIIz buffer: file name to look for, if found, changes to file path and name on exit

The buffer containing the string must be large enough to accommodate the resultant path name: approximately 255 characters.

## 4.2 Procedure FileName\_Filter

This call provides the “filter” functionality of ISYS, allowing either the current result-set or subsequent queries to be filtered based on file name, category or file date. The fields in the Filter Block correspond to the fields in the filter dialog of ISYS Query. Please see the ISYS online User’s manual for those definitions.

<u>Parameters</u>	<u>Description</u>									
wAccess	Use these constants from the header files: <table><tr><th><u>Constant</u></th><th><u>Value</u></th><th><u>Definition</u></th></tr><tr><td>ISYS_Filter_Next</td><td>&amp;hF0</td><td>filter subsequent queries.</td></tr><tr><td>ISYS_Filter_This</td><td>&amp;hFF</td><td>filter the current result-set.</td></tr></table>	<u>Constant</u>	<u>Value</u>	<u>Definition</u>	ISYS_Filter_Next	&hF0	filter subsequent queries.	ISYS_Filter_This	&hFF	filter the current result-set.
<u>Constant</u>	<u>Value</u>	<u>Definition</u>								
ISYS_Filter_Next	&hF0	filter subsequent queries.								
ISYS_Filter_This	&hFF	filter the current result-set.								
FilterBlock	Pointer to a Filter_Block									
QRB	Pointer to a Query_Result_Block: see <i>Perform_Find</i>									
EmsgBlk	Pointer to Error_Control_Block									

**Filter\_Block**

Path_Contains	255-character ASCIIz string: path contains this value
Path_Omits	255-character ASCIIz string: path does not contain this value
Fname_Like	255-character ASCIIz string: file name is like this value
Fname_Unlike	255-character ASCIIz string: file name is not like this value
Category_Like	255-character ASCIIz string: category name is like this value
Category_Unlike	255-character ASCIIz string: category name is not like this
Date_Before_Year	Smallint: four digit year. Document changed time stamp
Date_Before_Month	Smallint: in the range 1–12
Date_Before_Day	Smallint: in the range 1–31
Date_After_Year	Smallint: four digit year
Date_After_Month	Smallint: in the range 1–12
Date_After_Day	Smallint: in the range 1–31
DateInDoc_Before_Year	Smallint: four digit year. First date in the document.
DateInDoc_Before_Month	Smallint: in the range 1–12
DateInDoc_Before_Day	Smallint: in the range 1–31
DateInDoc_After_Year	Smallint: four digit year
DateInDoc_After_Month	Smallint: in the range 1–12
DateInDoc_After_Day	Smallint: in the range 1–31
DateIndexed_Before_Year	Smallint: four digit year. Date document was indexed.
DateIndexed_Before_Month	Smallint: in the range 1–12
DateIndexed_Before_Day	Smallint: in the range 1–31
DateIndexed_After_Year	Smallint: four digit year
DateIndexed_After_Month	Smallint: in the range 1–12
DateIndexed_After_Day	Smallint: in the range 1–31
File Type	Longint: a file format code from Appendix A

In Filter\_Block, if a field is to be blank to indicate “Don’t Care,” set the field equal to a constant from the header files. Use these constants:

<u>Constant</u>	<u>Value</u>	<u>Definition</u>
ISYS_Filter_DontCareName	"" (empty string)	Don’t care what is in the string (For path and name parameters)
ISYS_Filter_DontCareDate	0	Don’t care what date. (For data parameters)

You can use the DOS standard wild cards \* and ? in the Filename\_Filter call for the Fname\_Like field. You can use Fname\_Like = “\*.TXT” or Fname\_Unlike = “PART????.\*”. File name and path filtering is performed on document name as recorded in the ISYS index.

If the filter is applied to the current result-set, the QRB is filled with the effective query results after the call to FileName\_Filter. You may reapply different filters (or a null filter) any number of times to the current result-set. It is not necessary to re-execute the query to clear the effects of a filter imposed after the query has taken place.

**Code Example:**

```
wAccess = ISYS_Filter_Next

On Error Resume Next

'Text1..Text10 are text fields in the dialog box form.
FilterBlock.Path_Contains = String_to_AscIiz(text1)
FilterBlock.Path_Omits = String_to_AscIiz(text2)
FilterBlock.Fname_Like = String_to_AscIiz(text3)
FilterBlock.Fname_Unlike = String_to_AscIiz(text4)
FilterBlock.Date_Before_Year = CInt(text5)
FilterBlock.Date_Before_Month = CInt(text6)
FilterBlock.Date_Before_Day = CInt(text7)
FilterBlock.Date_After_Year = CInt(text8)
FilterBlock.Date_After_Month = CInt(text9)
FilterBlock.Date_After_Day = CInt(text10)

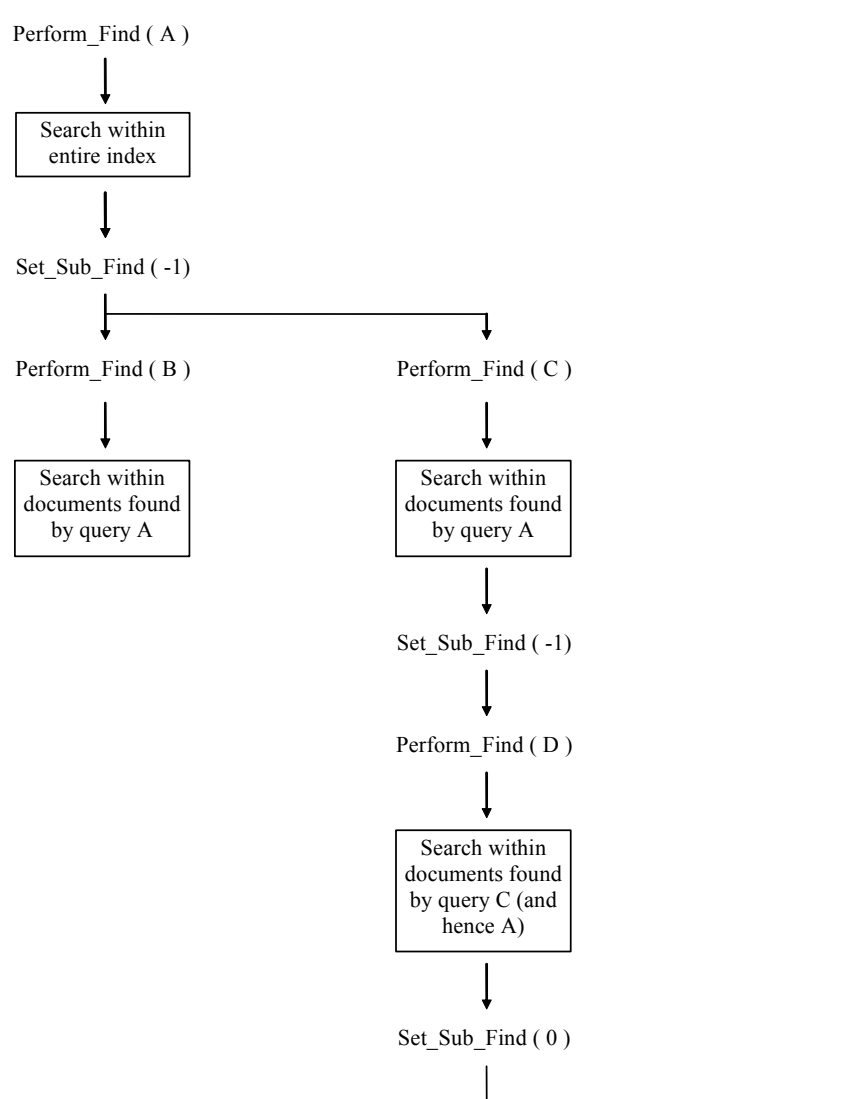
Call  Filename_Filter(wAccess,  FilterBlock,  MyQueryResult,
ISYS_Error)
```



### 4.3 Procedure Set\_Sub\_Find

When a result-set is in effect, calling Set\_Sub\_Find with a non-zero parameter means that all subsequent queries executed will be resolved within the result-set that was in effect. This situation remains in effect until Set\_Sub\_Find is called with a zero parameter. Having made a search, you may want to conduct a further search within the results of the current one.

All subsequent queries issued take place within the search results of the outer query, until such time that you call Set\_Sub\_Find with a zero parameter. To drill further down, call Set\_Sub\_Find with a non-zero parameter again. After drilling down, calling Set\_Sub\_Find with a zero parameter takes you all the way back to the top, not just back one step. The following diagram illustrates this behavior:



<u>Parameters</u>	<u>Description</u>
wBoolean	Word Boolean: zero to clear the sub-find or non-zero to establish the current result-set as the sub-find
EmsgBlk	Pointer to Error_Control_Block

Note that using the ISYS\_Query\_AND\_With\_Current flag with Perform\_Find offers another way of achieving a similar outcome, and has the advantage of preserving hits in both the original and subsequent queries.

## 4.4 Procedure Set\_Rlist\_Curfew

You may wish to limit the resources that can be expended evaluating a query, either because of memory restrictions or because of time limits. Set\_Rlist\_Curfew lets you impose a limit on how many resources will be expended resolving an ISYS query expression.

If a query is being processed and the number of hits found for a particular word exceeds this limit, evaluation of the query will be terminated immediately and the error “OEM defined retrieve list curfew” will be returned by Perform\_Find in the EmsgBlk parameter. Your application may detect the text of this message and take whatever action is appropriate.

When the curfew is tripped, the query is considered to be in an error condition, and no result set is returned.

This curfew applies for individual words, not necessarily the result of a query. Thus, if you set a retrieve list curfew of 10,000 references and a query is entered for the phrase “John Smith,” the curfew will be tripped if either the word “John” or the word “Smith” occurs more than 10,000 times. The limit is irrespective of the number of times they occur as a phrase. The application may, if you choose, reset the curfew and resubmit the query.

<u>Parameters</u>	<u>Description</u>
ListLimit	Longint: contains new limit; zero removes any limitation
EmsgBlk	Pointer to Error_Control_Block

## 4.5 Procedure Set\_Maximum\_Found\_Documents

Whereas Set\_Rlist\_Curfew abandons query evaluation when the resource limit is reached, Set\_Maximum\_Found\_Document allows a query to produce a coherent result set, but ensures that it never contains more than the nominated number of documents. This is similar to the way some Internet search engines behave in order to produce a result which, while may not be complete, is consistent within itself.

This is a useful way of imposing resource constraints, while still ensuring users get a result which is of use.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
DocLimit	Longint: contains new limit; zero removes any limitation
EmsgBlk	Pointer to Error_Control_Block

## 4.6 Procedure Word\_Search

This call provides the ability to search the database vocabulary, either on a word-stem or a “sounds like” basis. Word\_Search keeps track of its position in the search by passing a context block back and forth between the calling application and the DLL. Your application needs to define the block and pass it to Word\_Search on every call. For the first call, set the first four bytes of the context block to zero. This differentiates between “find first word that matches” and “find next word that matches”.

Note that where a chain of indexes has been opened at once, the Word\_Search function only operates on the first index in the chain.

<u>Parameters</u>	<u>Description</u>									
TargetPstr	Pointer to an ASCIIz string: either a word prefix, in the case of a stem search, or a word, in the case of a “sounds like” search.									
wSound	Use these constants from the header files: <table><tr><th><u>Constant</u></th><th><u>Value</u></th><th><u>Definition</u></th></tr><tr><td>ISYS_Wsearch_Stem</td><td>0</td><td>perform a stem search</td></tr><tr><td>ISYS_Wsearch_Sound</td><td>1</td><td>perform a sounds-like search</td></tr></table>	<u>Constant</u>	<u>Value</u>	<u>Definition</u>	ISYS_Wsearch_Stem	0	perform a stem search	ISYS_Wsearch_Sound	1	perform a sounds-like search
<u>Constant</u>	<u>Value</u>	<u>Definition</u>								
ISYS_Wsearch_Stem	0	perform a stem search								
ISYS_Wsearch_Sound	1	perform a sounds-like search								
AnswerPstr	Pointer to an ASCIIz string: client-supplied buffer that will hold a representation of the word found which matches TargetPstr or to a NULL word if no (more) matches were found									
pLoccs	Pointer to a Longint: will be set to the number of occurrences of the AnswerPstr word that exists in the database									
pContext	Pointer to Word_Search_Context: sets the Link field to 0 for the first call to Word_Search									
EmsgBlk	Pointer to Error Control Block									

### **Word\_Search\_Context**

Link	Longint: set to 0 for first call to Word_Search
Reserved	124-byte area used by Word_Search

## 4.7 Procedure Remove\_Document\_Record

Removes a document entry from the current list. In the ISYS:desktop product, this is called when the user presses the DEL key while looking at a document list.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
Which	Longint: document handle of document to remove from list
EmsgBlk	Pointer to Error_Control_Block

## 4.8 Procedure Interrupt\_Processing

During lengthy processing, the client application may call this entry point to signify to the Engine that it should abandon the current processing before completed. Interrupt\_Processing returns immediately and the DLL will end the current process in an orderly fashion soon thereafter.

The DLL never yields control of the processor or processes any Windows messages: it makes no assumptions about how you want your application to handle cooperative multitasking. The DLL does send status messages back to the window you specify in the call to Init\_Instance; this allows your application to process your own Stop or Cancel events and to process other messages.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
EmsgBlk	Pointer to Error_Control_Block

## 4.9 Procedure Get\_Hit\_Pointer

This call lets your application access low-level information about the query. It is supplied for users with more advanced requirements. It returns the document/paragraph/word pointer that ISYS uses internally.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
Refno	Longint: "hit number" in the range 1 to QRB.Total_Hits
RefPtr	Pointer to a Word_Pointer_Block
EmsgBlk	Pointer to Error_Control_Block

**Word\_Pointer\_Block**

Block that contains the A-P-W (Article-Paragraph-Word) location of a specific word in a result-list.

Word\_Num Word: contains the word number in the paragraph

Paragraph Word: contains the paragraph number in the document

Document Longint: contains the number of the document

Term\_Id Word: term id of this hit. Call Get\_Term\_ID\_Term to map the id into an actual search term from the query

**4.10 Procedure Get\_Document\_Type**

Deprecated. Use Get\_Document\_Record instead.

**4.11 Function****Get\_Document\_Fully\_Qualified\_FileName**

Deprecated. Use Get\_Document\_Record instead.

**4.12 Function Get\_Document\_Long\_FileName**

Deprecated. Use Get\_Document\_Record instead.

**4.13 Procedure Get\_Format\_Attribute**

Given a format code, returns its text description.

<u>Parameters</u>	<u>Description</u>
Num	Smallint: contains the document format code. See Appendix A for a complete list.
Format	Smallint: determines the format of the returned string. Use these constants:

<u>Constant</u>	<u>Value</u>	<u>Definition</u>
ISYS_Format_Name	1	descriptive name
ISYS_Format_Code	2	code used in ISYS.CFG

Result	Pointer to ASCIIz string: contains the result
EmsgBlk	Pointer to Error_Control_Block

## 4.14 Function Get\_Document\_Time\_Stamp

Given a reference to an open document, Get\_Document\_Time\_Stamp returns a Longint “time stamp” that may be used to check if the document has changed since it was indexed. The stamp returned is the “current” stamp, not the stamp when indexed. For most file formats, this is equivalent to the operating system file time stamp, but for other formats, it may be a symbolic generation identifier.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
hDoc	Longint: document reference as returned by Open_Document
EmsgBlk	Pointer to Error_Control_Block
<b>Returns</b>	Longint: time stamp

The format of a 32-bit operating system time stamp is as described here:

Two 16-bit values (least significant byte first)

<b><u>Element</u></b>	<b><u>Bits Used</u></b>	<b><u>Values</u></b>
Day	0-4	1-31
Month	5-8	1-12
Year	9-15	0-119 (year biased by 1980)
Seconds	0-4	0-29 (multiply by 2 to get seconds)
Minutes	5-10	0-60
Hours	11-15	0-24

## 4.15 Function Get\_Document\_Line\_Page\_No

Returns the “page” number of a given line. For file formats that internally record page numbers, ISYS attempts to follow those page numbers. For ASCII files, ISYS uses the chr(12) (form feed) character to determine page numbers. ISYS does not insert page breaks – page breaks must have been created by the originating application.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
hDoc	Longint: reference returned when the document was opened
LineNum	Longint: in the range 1 to NumLines (returned by Open_Document)
EmsgBlk	Pointer to Error_Control_Block
<b>Returns</b>	Word page number.

## 4.16 Procedure Get\_EAM\_Context\_Info\_for\_Hit

Given a handle to a document that you previously opened using Open\_Document and a hit number from 1 to the number of hits in that document, returns a 12-byte context area that corresponds to the first character of where that hit occurs.

This is useful where External Access Modules are involved, as the 12-byte context area is exactly that which the EAM returned to ISYS when the document was originally read for indexing. This lets your application do its own highlighting if you want to provide your own “direct” viewing without using Get\_Document\_Line.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
hDoc	Longint: handle to an open document, as returned by Open_Document
HitNum	Longint: hit number, 1 to number of hits in document. Get_Document_Record tells you how many hits are in a particular document. If HitNum is out of range, Context will be set to zeros.
Context	Pointer to EAM_Context: fills the buffer with a value equal to the 12-byte context area your External Access Module returned when the first character of this “hit word” was returned.
EmsgBlk	Pointer to Error_Control_Block

### **EAM\_Context**

Info Array [1..12] of Byte: internal

The EAM\_Context could just as well be any other 12-byte data structure. See the External Access Modules section for more information.

## 4.17 Function Auto\_Determine\_File\_Format

If you need to find out the file format of a particular file, call Auto\_Determine\_File\_Format. ISYS uses a heuristic algorithm; it may open the file and read the first 2k in order to determine the result. In some cases, documents contain an unambiguous signature which positively identifies their format. In other cases, ISYS will attempt to resolve ambiguous formats heuristically. The extension of the file format is usually not significant, but may in some cases be used as an adjunct to help resolve ambiguity.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
FileName	Pointer to an ASCIIz string: specifies the name of the file whose format is to be determined. This file need not previously have been indexed. The file passed is completely arbitrary.
EmsgBlk	Pointer to Error_Control_Block
<b>Returns</b>	Smallint file format code.

The function returns an integer value representing the format of the file. The format code will be one of the integer values listed in Appendix A, or may be one of these values:

<b><u>Value</u></b>	<b><u>Definition</u></b>
0	Is not one of the known formats.
1	Is a recognized format, but not one ISYS can index.
2-81	Is a recognized file format. See Appendix A.
Other	File does not exist or cannot be opened.

ISYS makes a “best guess” of the file format based on the known file formats. This call is not meaningful for information types that do not correspond to operating system files. For example, the call makes sense for spreadsheets, word processor files, ASCII files, HTML files, and so forth, but does not make sense for files that are of a proprietary format (accessed through an External Access Module), or email or SQL data where the physical operating system file is a *container*, not the information *itself*, and requires the services of a server application to open the container and reveal the information within.

## 4.18 Procedure Translate\_Hyperstring

This routine provides a way to use a section of retrieved text as a further query. Translate\_Hyperstring accepts a pointer to a buffer that contains an ASCIIz string of any arbitrary text. It removes all common words, conflates the rest, removes duplicate words, and links the remaining words with “within a paragraph of” operators. The resultant string constitutes a valid ISYS query command that you can then resubmit to ISYS though the Perform\_Find command.



Also see Find\_Similar, which provides a more sophisticated means of finding text which is similar to other text.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
Pstr	Pointer to a buffer containing an ASCIIz string
EmsgBlk	Pointer to Error_Control_Block

## 4.19 Function Open\_Unfound\_Document

When you need to open a document which has not been located as the result of a search, call Open\_Unfound\_Document. One example of such a document is a static hypertext linkage. The function is analogous to Open\_Document, except that instead of quoting a document number within a current retrieved set, you quote the actual name of the document.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
Name	Pointer to ASCIIz string containing the name of the file to open.
Num_Lines	Pointer to Longint: gets the number of lines in the document.
Doctype	Pointer to Smallint: gets the ISYS type code for the document. See Appendix A: Document Format Codes.
FnameStr	Pointer to an 255-character ASCIIz buffer: Open_Unfound_Document fills this with the fully qualified physical file name of the document as it is actually opened. With a ZIPped document, this may be the name of a temporary file.
EmsgBlk	Pointer to Error_Control_Block
<b>Returns</b>	Smallint: document reference number

Note that an ISYS index must be open at the time you make this call.

If the document has been indexed, ISYS will use the document type and options that are in effect within the index, but the name you quote must be exactly as it appears in the index. If the Name is not an indexed document, then ISYS will automatically determine the file format from among the formats available in the FORMATS statement in the ISYS.CFG file.

Note that if you have documents indexed in WYSIWYG mode, and you call Open\_Unfound\_Document correctly passing the document name as it is recorded in the index, then ISYS will open the document in WYSIWYG mode. However, if you passed the file name in a form that is not identical to how it is stored in the index, but is still a valid way of referring to the same file, then ISYS will assume you are opening a non-indexed document and will open it in non-WYSIWYG mode.

The function returns a reference to the open document. If the document could not be opened due to an open error, it returns ISYS\_Open\_Failed (-1).

## 4.20 Procedure Get\_Term\_ID\_Term

Maps a query term id into an actual query term. Query terms are identified with numbers starting contiguously from 48, and the decoded terms are words that appeared in the query. If you request a term outside the valid range, a null string is returned. Thus to enumerate all the terms used in a query, call this routine in a loop starting with TermId 48 until an empty term string is returned.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
TermId	Longint: the ID of the term to be decoded.
Term	Pointer to a caller-supplied area of memory to be filled with an ASCIIz representation of the query term identified by TermId.
EmsgBlk	Pointer to Error_Control_Block

## 4.21 Function Get\_Line\_With\_Term\_After

## 4.22 Function Get\_Line\_With\_Term\_Before

These routines are analogous to Get\_Line\_With\_Hit\_After and Get\_Line\_With\_Hit\_Before, except allow direct navigation from one specific term to another. For example, if your query was for Dog and Cat, these calls let you navigate directly from Dog to Dog, or Cat to Cat. This form of navigation is only available for non-WYSIWYG documents, or WYSIWYG documents opened in non-WYSIWYG mode.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
hDoc	Longint: handle to an open document, as returned by Open_Document
TermID	Longint: the ID of the term to be located, as determined from Get_Term_Id_Term
Line_No	Longint: the line number after which, or before which, to locate the term
EmsgBlk	Pointer to Error_Control_Block
<b>Returns</b>	Longint: line number, or zero if no such line exists

## 4.23 Procedure Get\_Category\_Record

ISYS includes the ability to automatically assign documents to categories. This assignment occurs according to rules. The rules are automatically generated by ISYS, but may also be augmented by your own rules. More information is available in the ISYS Utilities help file.

This routine lets you determine what categories were found by the current query. Call this routine requesting category number 1, and continue calling passing sequentially higher category numbers until an empty category name is returned.

<u>Parameters</u>	<u>Description</u>
CatNum	Longint: the category number to fetch
CatRec	Pointer to a caller supplied Category_Control_Block
EmsgBlk	Pointer to Error_Control_Block

### Category\_Control\_Block

Block that contains the details of a category found by the current query.

Name	ASCIIz: category name (100 bytes)
Num_Docs	Longint: number of documents found in this category
Relevance	Longint: unweighted average relevance of documents found in this category
Weight	Longint: weighing which should be applied to this category

## 4.24 Procedure Find\_Similar

Given a portion of text, this routine will attempt to find other documents which match the key content of the text. ISYS will use linguistic tense conflation, synonyms and statistical methods to find documents which best match. Documents found by this call replace the current result set.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
Text	Pointer to an ASCIIz string containing the text to which you would like to find similar documents
hWnd	Handle of a window which is to receive notification messages as the query proceeds. If no such notifications are required, this may be zero.
QRB	Pointer to a caller-allocated Query_Result_Block which will be filled by ISYS
EmsgBlk	Pointer to Error_Control_Block

## 4.25 Procedure Get\_Document\_Group\_Entry

ISYS has the notion of documents existing in associated storage groups. Examples of this include an email plus its attachments, the various entries of a ZIP file, and an SQL or Lotus Notes record and all of the BLOB documents held in that record.

This routine lets you enumerate the documents which are in the same storage group as a given document.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
BaseDocName	Pointer to an ASCIIz string containing the name of the document for which you wish to retrieve all the documents in the same storage group.
EntryNum	Longint: used to enumerate the documents in the storage group. Call passing in an EntryNum of 1, and continue calling incrementing the EntryNum until the GroupEntryName returns null.
GroupEntryName	Pointer to a caller-allocated buffer of at least 256 bytes. Will be filled by the DLL with the ASCIIz document name of that group entry number.
EmsgBlk	Pointer to Error_Control_Block

## 4.26 Procedure Get\_Query\_Spelling\_Suggestion

If your index has the generation of spelling suggestions enabled in the ISYS.CFG, then this call may be used to retrieve a list of alternate spellings for any words in the query for which ISYS determines alternates are warranted.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
Query	Pointer to an ASCIIz string containing the text of the query most recently executed.
EntryNum	Longint: enumerator to control which spelling suggestion you wish to receive. Start by requesting entry 1, and continue until Start_Word_Offset returns zero.
Start_Word_offset	Pointer to a Longint: returns the starting position of the word for which alternate spellings are being suggested. The first character of Query is numbered 1.
Word_Length	Pointer to a Longint: returns the length of the word in Query for which alternate spellings are being suggested.
Suggestions	Pointer to a caller-supplied buffer of at least 256 characters in length which will be filled by the DLL with a list of alternate spellings. Each alternate is space-delimited, and they are provided in order of preference.
EmsgBlk	Pointer to Error_Control_Block

## 4.27 Procedure Get\_Document\_Metadata

If you have metadata caching enabled for your index, you may use this routine to access the metadata for selected documents from your results list, without first opening the document.

Metadata items are requested by case-insensitive item name, or if a null name is passed in, this routine will return a list of all the available metadata items. A maximum of 6k is returned.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
DocNo	Longint: document number in the range 1 to QRB.Num_Docs.
Item	Pointer to an ASCIIz string containing the name of the metadata item required. May be a null pointer or a pointer to a null string to return the names of all the metadata items available for this document.
Result	Pointer to a caller-supplied buffer of at least 6k in length which will be filled by the DLL with the ASCIIz value of the metadata item requested. If you have requested a list of all the metadata items, this buffer will be filled with a list of ASCIIz metadata names, each terminated by a null. The list itself will be terminated by a double null.
EmsgBlk	Pointer to Error_Control_Block

## 4.28 Procedure Set\_Metadata\_Relevance\_Boost

ISYS relevance ranking is based on a number of inter-relating factors. If all other factors were equal (which is rarely the case), ISYS would give preference to hits occurring in metadata, but it is also very possible that a preponderance of hits in the body of a document may cause it to be ranked ahead of a document with only a single hit occurring in the metadata.

This routine allows you to control to what extent hits in metadata are “preferred” from a relevance ranking aspect.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
Amount	Longint: amount of relevance “boost” to apply to hits occurring in metadata. Ranges from 0 (default) to 10. When set to maximum, even the least relevant hit in metadata will be considered more relevant than the most relevant hit in non-metadata.
EmsgBlk	Pointer to Error_Control_Block

## 4.29 Procedure Set\_Metadata\_Query\_Scope

ISYS queries apply to both metadata and document bodies, unless specifically limited using the IN or ESPIN operators, in which case the query is limited just to the field or labeled paragraph specified, or unless the METAONLY or BODYONLY query operators are used. This routine may be used to limit searching to either just the metadata portion (all fields) or just the non-metadata portion of documents.

<u>Parameters</u>	<u>Description</u>
Flags	Longint: determines which portion of documents will be included in subsequent queries:
<u>Constant</u>	<u>Value</u> <u>Definition</u>
ISYS_MetaScope_All	0 Both portions
ISYS_MetaScope_Meta_Only	1 Metadata only
ISYS_MetaScope_Body_Only	2 Non-metadata only
EmsgBlk	Pointer to Error_Control_Block

### 4.30 Procedure Set\_Metadata\_Sort\_Key

If calling Sort\_Documents with a sort method of ISYS\_Sort\_Metadata, this routine determines which metadata field the documents should be sorted by. Metadata items are identified by name. It is understood that not all documents will contain the same set of metadata, and any documents which omit the metadata field specified will be sorted to the end of the sequence. In other words, null collates high.

<u>Parameters</u>	<u>Description</u>
Item	Pointer to an ASCIIz string containing the name of the metadata field by which to sort.
EmsgBlk	Pointer to Error_Control_Block

### 4.31 Procedure Close\_Find

Closes a query and discards the result-set. Any open documents associated with that query will remain open and may continue to be used. The DLL performs an implied Close\_Find when a new query is executed or when a different database is selected.

<u>Parameters</u>	<u>Description</u>
EmsgBlk	Pointer to Error_Control_Block

## 4.32 Procedure Close\_Database

Closes the database currently in use and clears the current result-set. Documents from the old database **may** be left open if required.

<u>Parameters</u>	<u>Description</u>
EmsgBlk	Pointer to Error_Control_Block

## 4.33 Performing Fielded Searching

Many SDK users have applications with structured data, be it SQL or proprietary, or even just structured information in ASCII files.

Traditional SQL databases are very good at structured searching within fields, and very poor at unstructured searching. This is why people license text search engines such as ISYS. For example, SQL databases are excellent at finding an exact value in a particular column or field. They are terrible at finding a word embedded somewhere inside all of the columns in a table.

In other words, structured SQL systems are good at:

```
SELECT MyKeyField
FROM MyTable
Where MyField = 'ME'
```

And very poor at:

```
SELECT MyKeyField
FROM MyTable
Where (Myfield01 LIKE '* ME *')
      OR (Myfield02 LIKE '* ME *')
      OR (Myfield04 LIKE '* ME *')
      OR (Myfield05 LIKE '* ME *')
```

Text search engines have the inverse property. They work best when searching across all “fields” because this is what they are designed to do.

For comparison, a similar ISYS query for the second SQL statement above would be simply:

```
ME
```

and for the first SQL statement would be

```
ME IN MyField
```



The situation becomes even more pronounced where you are searching across multiple tables. In the SQL case, a UNION operator would be required, and the net effect would be doing a series of full table scans across multiple tables - a very expensive procedure. In the ISYS case, the query remains the same and the performance remains proportional *to the size the result will be*, not the amount of data you have.

The corollary is to use your SQL tools where SQL-style searching is more efficient, and ISYS searching where ISYS searching is most efficient. Yet often it can be extremely convenient to perform fielded searching in ISYS as well. There are two ways this can be done: simple and flexible, or premeditated and highly efficient.

At the simplest level, the ISYS “IN” operator is the basis of fielded searching. It lets you do queries like this:

(Cat <b>in</b> MyField01) and (Dog <b>in</b> MyField02)
---

You don't need any special preparation or mark-up to make it happen - it works automatically based on your information content. ISYS field searching even allows a great deal of flexibility in dealing with data that doesn't conform to strict relational rules, for example, records can contain their 'fields' in a different order to each other, 'fields' can occur more than once, or even not at all.

Within the framework of a result of a constant size, it may seem counterintuitive that an unfielded search is actually much more efficient than a fielded search.

This is because simple fielded searching comes at a performance cost, as its first task is to find where the 'field' exists in each record before applying the search. (The 'field' is just a paragraph that begins with a particular word or phrase).

The process of locating where the *fields* occur may be more resource consumptive than locating the target words themselves, because there may be many more occurrences of the field label than the target word, and ISYS search performance is related to result size. For example, there might be 100,000 “MYFIELD01”s, but only 100 “CAT”s in those fields.

In other words, the flexibility of the “IN” operator comes at a performance cost which may be substantial.

Moreover, for heavily structured SQL data, the initial step of locating where the chosen field is in each record is redundant because your data *does* follow strict relational rules and the fields are in the same position in each row of the table.

You can use this knowledge to leverage a huge performance boost on fielded searching by specifying the field by ordinal position instead of name. For example:

<b>Field 2</b> (CAT)
----------------------

This form of search performs the sub-query in the parenthesis only within the second field (paragraph) in each and every record. You can combine ordinal field searching and unrestricted searching in the same query:

<code>(<b>field</b> 2 (CAT)) and VET</code>
---

The easiest way to know the ordinal value of your field is to perform a query using ISYS:desktop, open a record, and look for the paragraph markers on the left hand side of the document rendition of your SQL record. Your application needs to generate the search expression with the appropriate ordinal field number based on whatever user interface paradigm you have selected - often a fill-in-the-form approach to fielded searching maps very easily into ordinal based sub-queries.

Although ordinal-based field searching does involve a little more preparatory effort than a simple “IN” search, the performance difference can be breathtaking.

---

## 5. UTILITIES

<u>API Routine</u>	<u>Description</u>
CFG_Settings	Gets or sets parts of the ISYS.CFG file.
CFG_Formats	Gets or sets file format information in ISYS.CFG file.
Def_Settings	Gets or sets attributes from the ISYS.SET file.
Open_Rules	Initializes index rules for editing.
Fetch_Rule_Areas	Gets list of directories and file-association rules.
Fetch_Rule_List	Gets rule list for the current directory.
Create_Rule	Adds a file-association rule to the current directory.
Move_Rule	Moves a rule.
Close_Rules	Finishes a rule-editing session.

Unlike all query-oriented API calls, the utility calls must take place without a database being opened. They open, process, and close the database as required. They operate on the ISYS database in the current directory, therefore, set the current directory in your program before making these calls.

These calls do not have to be made to modify the ISYS.CFG file. The ISYS.CFG file is just an ASCII file that can be opened or created as a normal text file and these API calls are added should the programmer desire to access the file in a way that is guaranteed to be compatible with other ISYS applications accessing the same file.

Usually, most applications will make use of a pre-defined ISYS.CFG file with specific needs, in which case it is far easier for the application to just emit the ISYS.CFG file as an ASCII file, or prepare the ISYS.CFG file manually and ship and install it with your application.

Note: for most applications, it is simplest and easiest to manually preconfigure your ISYS.CFG file and ship it with your application, than it is to generate it programmatically via this API.

## 5.1 Procedure CFG\_Settings

Gets or sets the non-rule, non-format options from the ISYS.CFG file in current directory.

<u>Parameters</u>	<u>Description</u>
CFGBlock	Pointer to a Configuration_Control_Block
EmsgBlk	Pointer to Error_Control_Block

### Configuration\_Control\_Block

Action	Character: use a constant from the header files:									
	<table><tr><th><u>Constant</u></th><th><u>Value</u></th><th><u>Definition</u></th></tr><tr><td>ISYS_Load</td><td>“L”</td><td>load the structure from CFG file</td></tr><tr><td>ISYS_Save</td><td>“S”</td><td>save the structure to CFG file</td></tr></table>	<u>Constant</u>	<u>Value</u>	<u>Definition</u>	ISYS_Load	“L”	load the structure from CFG file	ISYS_Save	“S”	save the structure to CFG file
<u>Constant</u>	<u>Value</u>	<u>Definition</u>								
ISYS_Load	“L”	load the structure from CFG file								
ISYS_Save	“S”	save the structure to CFG file								
Name	255-character ASCIIz: database name									
Concurrency	Boolean word: indicates database concurrency									
Numbers_Common	Boolean word: indicates that numbers are common									
Latency	Smallint: number of minutes to wait after a file has been saved before beginning to index it.									
Signif	255-character ASCIIz: significant characters. List of additional characters which are to be considered part of a word (a-z, 0-9 are assumed). For ANSI characters, specify any additional 8-bit characters. For Unicode indexes, only specify characters in the range 1-127. Characters beyond the 7-bit range are interpreted according to operating system specifications.									
InSignif	255-character ASCIIz: <i>insignificant</i> characters. List of additional characters which are considered part of a word, but not significant enough to distinguish one word from another. ANSI/Unicode rules are as above.									
AnnoDrive	255-character ASCIIz: AnnotationDrive, blank for default									
Title_Line	Smallint: title line number									
Title String	100-character ASCIIz: title line contains									

Date_Handling	Boolean word: indicates whether intelligent date handling is on																																
Dot_Handling	Boolean word: indicates how periods are to be handled																																
Filename_Indexing	Boolean word: indicates whether filenames are to be indexed																																
Annotation_Indexing	Boolean word: indicates whether annotations are to be indexed																																
Master_Doc	255-character ASCIIz: path and file name of master document																																
Ocr_Precomp	Boolean word: indicates whether OCR precompensation is on																																
Number_Handling	Boolean word: indicates if intelligent number handling is on																																
Doc_Profiles	Boolean word: indicates if Word summary information is indexed																																
LanguageCode	<p>Smallint: the special language interpretation options for this index:</p> <table> <tr><td>0</td><td>None</td></tr> <tr><td>1</td><td>Ignore accents</td></tr> <tr><td>2</td><td>Korean</td></tr> <tr><td>3</td><td>Traditional Chinese</td></tr> <tr><td>4</td><td>Hong Kong Chinese</td></tr> <tr><td>5</td><td>Japanese</td></tr> <tr><td>6</td><td>Arabic</td></tr> <tr><td>7</td><td>Simplified Chinese</td></tr> <tr><td>8</td><td>Cyrillic</td></tr> <tr><td>9</td><td>Greek</td></tr> <tr><td>10</td><td>Turkish</td></tr> <tr><td>11</td><td>Hebrew</td></tr> <tr><td>12</td><td>Vietnamese</td></tr> <tr><td>13</td><td>Baltic</td></tr> <tr><td>14</td><td>Unicode</td></tr> <tr><td>15</td><td>Central European</td></tr> </table> <p>Indexes which use language code 14 normalize their content into the Unicode character set, and are able to contain multiple disparate languages simultaneously.</p>	0	None	1	Ignore accents	2	Korean	3	Traditional Chinese	4	Hong Kong Chinese	5	Japanese	6	Arabic	7	Simplified Chinese	8	Cyrillic	9	Greek	10	Turkish	11	Hebrew	12	Vietnamese	13	Baltic	14	Unicode	15	Central European
0	None																																
1	Ignore accents																																
2	Korean																																
3	Traditional Chinese																																
4	Hong Kong Chinese																																
5	Japanese																																
6	Arabic																																
7	Simplified Chinese																																
8	Cyrillic																																
9	Greek																																
10	Turkish																																
11	Hebrew																																
12	Vietnamese																																
13	Baltic																																
14	Unicode																																
15	Central European																																
Meta_Titles	Boolean word: indicates whether document titles are taken from the document meta data, for formats that support meta data titles.																																
Flush_Index_n_Docs	Longint: during an update run, this setting causes the work files to be flushed into the index after a certain number of documents. If this value is set to zero, ISYS will flush as rarely as available resources allow. It is most efficient to flush as rarely as possible, but you may choose to set this to a lower value in some circumstances.																																
Flush_Index_n_Words	Longint: similar to above, but forces a flush after a certain number of words.																																

Backup_Generations	SmallInt: determines how many generations of automatic index backup ISYS will generate. A backup generation is taken every time the index is opened with write intentions.
Default_File_Options	SmallInt: controls what indexing options to apply by default to indexing rules, and also apply to non-filesystem-based documents, e.g. attachments to emails or BLOBs in SQL databases. This value is set in the same way as Set_Concord_From_File_Option.
Max_Word_Length words	Wordint: controls the maximum significant indexed word length in bytes. Defaults to 20 for ANSI indexes, or 30 for Unicode indexes (since Unicode is represented using UTF8 encoding, may be longer as measured in bytes). Absolute maximum is 64.
Load_EAM	255-character ASCIIz: name of External Access Module to load, if not ISYSEX32.DLL.
NonUnicode_Codepage	Word: controls what codepage to assume when processing a non-Unicode document format (for example, a plain ASCII file) into a Unicode index.
Deferred_Deleted_Cache_Percent	Boolean word: controls to what extent deleted (or changed) documents may have their deindexing deferred until later for efficiency purposes.
Spelling_Tips_Avail	Boolean word: controls whether spelling tip information should be generated during indexing, allowing applications to expose a “Did You Mean?” feature via Get_Query_Spelling_Suggestion.
Cache_Metadata	Boolean word: indicates whether document metadata should be cached into the index so it may subsequently be accessed using Get_Document_Metadata.
Cache_Security	Boolean word: indicates whether document security information should be cached into the index. Where documents exist on a different server to the index, this option can greatly enhance performance when Enforce_Security_Visibility is also selected.
Index_Type	Byte: used to indicate special purpose indexes
Custom_Props	255-character ASCIIz: special purpose additional information
Entity_Handling	Boolean word: indicates whether automatic entity recognition is enabled.
Cache_Documents	30Boolean word: whether document content caching is enabled.

Full details on these parameters can be found in the ISYS Utilities online help, under *Technical Details*, then *Index Configuration File*.

## 5.2 Procedure CFG\_Formats

This API returns or sets the file format information from an ISYS.CFG file located in the current directory. CFG\_Formats must occur between an Open\_Rules and Close\_Rules, otherwise the ISYS.CFG file isn't accessed.

<u>Parameters</u>	<u>Description</u>
Formats	Pointer to a Word: value indicates the number of formats present in FormatBlk. If negative, this represents the maximum number of formats that the DLL may load into FormatBlk, and the DLL will reset the value to the actual number loaded. If positive, then FormatBlk is written back to disk. <i>Multiply your number by these constants.</i>
<u>Constant</u>	<u>Value</u> <u>Definition</u>
ISYS_CFG_Read	-1      maximum number of formats to read
ISYS_CFG_Write	1      write formats to disk
FormatBlk	Pointer to an array of Format_Rec
EmsgBlk	Pointer to Error Control Block

The way to use this routine is to call it once with Formats equal to the negative maximum number of entries your FormatBlk can hold. This will fill your FormatBlk with the names and mnemonics of all the formats ISYS can understand. Then alter the Nominated booleans within your FormatBlk to select and deselect formats as required. Finally, call CFG\_Formats again with Formats equal to the same value returned by the first call. This will update the ISYS.CFG file with your selections.

### Format\_Rec

Name	40 character ASCIIz: descriptive WP name
Code	20 character ASCIIz: ISYS.CFG keyword
Nominated	Smallint boolean: is format selected or not

You must do an ISYS\_CFG\_Read of the formats in from the ISYS.CFG file before you can write them back out the file.

## 5.3 Procedure Def\_Settings

DEF\_Settings reads or writes the ISYS.SET file, which contains feature access and auditing settings. It is located on disk in the same way as for ISYS:desktop. If the ISYS.SET file is in the ISYS directory, it applies to all databases or if it is in just the database directory, it applies only to that database.

<u>Parameters</u>	<u>Description</u>
SetBlk	Pointer to an Action_Control_Block
EmsgBlk	Pointer to Error_Control_Block

### Action\_Control\_Block

For further information on these fields, please refer to the ISYS Utilities online help, under *Security*, then *Restricting User Access*.

Action	Character: Use a constant from the header files:									
	<table><tr><th><u>Constant</u></th><th><u>Value</u></th><th><u>Definition</u></th></tr><tr><td>ISYS_Load</td><td>“L”</td><td>load the structure from SET file</td></tr><tr><td>ISYS_Save</td><td>“S”</td><td>save the structure to SET file</td></tr></table>	<u>Constant</u>	<u>Value</u>	<u>Definition</u>	ISYS_Load	“L”	load the structure from SET file	ISYS_Save	“S”	save the structure to SET file
<u>Constant</u>	<u>Value</u>	<u>Definition</u>								
ISYS_Load	“L”	load the structure from SET file								
ISYS_Save	“S”	save the structure to SET file								
Dir_Constraint	255 character ASCIIz string: directory constraint									
Pr_Line_Limit	Smallint: printer line limit (0 = no limit)									
DDA_Allowed	Smallint boolean: DDA enabled									
SynEdit_Allowed	Smallint boolean: Synonym editing enabled									
LibEdit_Allowed	Smallint boolean: Saved query library editing enabled									
AnnoEdit_Allowed	Smallint boolean: Annotation editing enabled									
Copy_Allowed	Smallint boolean: File copy enabled									
Audit_On	Smallint boolean: Auditing enabled									
Audit_UID	255 character ASCIIz string: user identification string									
Audit_Fname	255 character ASCIIz string: audit file name									
Audit_Errors	Smallint boolean: audit error messages									



Audit_Queries	Smallint boolean: audit queries
Audit_Results	Smallint boolean: audit results
Audit_Browse	Smallint boolean: audit documents browsed
Audit_Activate	Smallint boolean: audit documents activated
Audit_Annotate	Smallint boolean: audit annotation changes
Audit_Starts	Smallint boolean: audit program starts/stops
Audit_Loads	Smallint boolean: unused
Audit_Databases	Smallint boolean: audit databases
DirSelect_Allowed	Smallint boolean: allow user to select database
CatEdit_Allowed	Smallint boolean: allow user to edit catalog of databases
Print_Allowed	Smallint boolean: allow user to print selected text
Enforce_Security_Visibility	Smallint boolean: before returning from processing a query, scan the document result list and check that each document will be accessible for the current user. Documents that would not be accessible are transparently removed from the document list before Perform_Find (or any other query method) returns. Applying this setting may have a substantial effect on performance.

## 5.4 Procedure Open\_Rules

This call initiates a sequence of ISYS.CFG indexing rule editing. Upon the call to Open\_Rules, the ISYS.CFG file is read into memory. It is rewritten upon the call to Close\_Rules. All actions are upon the ISYS.CFG file in the current directory at the time of this call. You should not call CFG\_Settings between Open\_Rules and Close\_Rules.

<u>Parameters</u>	<u>Description</u>
EmsgBlk	Pointer to Error_Control_Block

## 5.5 Procedure Fetch\_Rule\_Areas

This call returns an array of “rule areas” that correspond to each available drive on the system plus the directory in which the ISYS.CFG file itself resides.

The SelectArea parameter performs two jobs in one. If passed in as zero, the function sets the parameter NumAreas to the currently selected area. If passed as a negative number, it causes that area to become the actively selected area for all subsequent operations. For example, passing a value of -2 would cause area number 2 to become the currently selected area.

<u>Parameters</u>	<u>Description</u>									
NumAreas	Pointer to an Smallint: number of 60 byte strings in parameter AreaArray. The caller sets this to the maximum number of entries that AreaArray can contain and the DLL resets it to the number of entries actually used.									
SelectArea	Pointer to an Smallint: use constants from the header files to determine how to select the area. <i>Multiply one of these constants with the area number to generate the SelectArea parameter.</i>									
<table><tr><th><u>Constant</u></th><th><u>Value</u></th><th><u>Definition</u></th></tr><tr><td>ISYS_Rule_ThisArea</td><td>0</td><td>Use this or current area.</td></tr><tr><td>ISYS_Rule_NextArea</td><td>-1</td><td>Use this area the next time.</td></tr></table>		<u>Constant</u>	<u>Value</u>	<u>Definition</u>	ISYS_Rule_ThisArea	0	Use this or current area.	ISYS_Rule_NextArea	-1	Use this area the next time.
<u>Constant</u>	<u>Value</u>	<u>Definition</u>								
ISYS_Rule_ThisArea	0	Use this or current area.								
ISYS_Rule_NextArea	-1	Use this area the next time.								
AreaArray	Pointer to a program-supplied array of 60 byte ASCIIz strings: will be populated to a limit of NumAreas. Each entry will contain the drive and directory, a tab, and the displayable ASCII character number of rules that exist in that area.									
EmsgBlk	Pointer to Error Control Block									

## 5.6 Procedure Fetch\_Rule\_List

Returns the current rule list for the currently selected area (as selected by Fetch\_Rule\_Areas). Fetch\_Rule\_List only operates within the currently selected area.

<u>Parameters</u>	<u>Description</u>
NumRules	Pointer to an Smallint: represents the maximum number of rules that can be loaded into RuleArray. Reset by the DLL to the actual number loaded.
RuleArray	Pointer to a program-supplied array of 255 byte ASCIIz strings: will be populated with the rules for the currently selected area.
EmsgBlk	Pointer to Error_Control_Block

## 5.7 Procedure Create\_Rule

Creates a rule in the currently selected area. The rule is expressed in the same syntax as is used in the ISYS.CFG file and is documented in the ISYS Utilities online help. Create\_Rule only operates within the currently selected area. Note that filenames which contain spaces should be enclosed in quotation marks according to normal Windows conventions.

<u>Parameters</u>	<u>Description</u>
RulePstr	Pointer to an ASCIIz string rule
EmsgBlk	Pointer to Error_Control_Block

## 5.8 Procedure Move\_Rule

Moves a rule from one position to another in the currently selected area. Rules within an area are applied in order; the first one that matches is the one that applies. Move\_Rule allows you to change the order in which rules are evaluated and applied. Move\_Rule only operates within the currently selected area.

<u>Parameters</u>	<u>Description</u>
nFrom	Smallint: position from which to move rule
nTo	Smallint: new position for the rule. If zero, the rule is deleted.
<u>Constant</u>	<u>Value</u> <u>Definition</u>
ISYS_Rule_Delete	0            Delete the rule.
EmsgBlk	Pointer to Error Control Block

## 5.9 Procedure Close\_Rules

Releases the memory associated with the current set of rules and optionally saves them back to disk.

<u>Parameters</u>	<u>Description</u>
Save	Boolean Smallint: non-zero if the rules are to be saved or zero otherwise
EmsgBlk	Pointer to Error_Control_Block



---

## 6. INDEX UTILITIES

This chapter describes Procedure IDB\_Function, which performs most of the index maintenance functions of the ISYS Utilities application.

### 6.1 Procedure IDB\_Function

Executes an “IDB” utility function. Commands are passed using the syntax described below. Progress notification messages are passed back via the Window handle nominated in the call to Init\_Instance. You may call Interrupt\_Processing to terminate processing before it reaches its natural conclusion. Interrupt\_Processing does not immediately terminate processing, but just signals to the DLL that it should stop when it reaches a convenient point.

All processing is performed on the index in the current directory. Therefore, within your application, change the current directory to the one in which your desired index resides before issuing an IDB\_Function call.

<u>Parameters</u>	<u>Description</u>
CmdPstr	Pointer to an ASCIIz string: contains the IDB Command to perform. See the following table for recognized strings.
EmsgBlk	Pointer to Error_Control_Block

<u>IDB Command</u>	<u>Description</u>
CREATE	Initializes an index and processes its documents.
UPDATE	Updates an index to match changes in documents.
OPTIMIZE	Processes the index for optimal space utilization.
REINDEX	Reinitializes an index and processes its documents.
PREVIEW	Scans documents for changes.
STATISTICS	Gathers and reports statistics for an index.
FREQ	Lists word frequency list for an index.
LIST	Lists all the documents in an index.
COMMON	Lists the common words in an index.

VERSION	Returns the complete ISYS version number.
USERS	Returns the number of active and licensed users.
SUPPORT	Provides a number of index integrity checks.
ADD	Adds a document to the index.
RECATEGORYIZE	Forces the document categories to be recalculated.
REFRESHSECURITY	Forces security information to be reached.
PROCESSCACHEDDELETES	Forces deferred deletions to be processed.

The header files contain constants that define these commands. Their names are always "ISYS\_IDB\_" followed by the command. For instance, ISYS\_IDB\_Create = "CREATE".

IDB\_Function opens its own files automatically; when you call IDB\_Function, no index should be open. If you have an ISYS index open when you want to call IDB\_Function, call procedure Close\_Database. You may reopen it using Open\_Database once IDB\_Function has finished.

Messages are directed to the hWnd specified in Init\_Instance, with a message type of hex \$100 by default, the wParam equal to the handle of a Windows global memory block that contains the message, and the lParam set to a pointer to that locked global memory block. The first byte of the memory block indicates the type of the message and the format of the remainder of the block. The message types are:

<b>Message</b>	<b>Value</b>	<b>Definition</b>
ISYS_MSG_Stream	"M"	Streaming message. The remainder of the block is an ASCIIz string that is to be displayed in a scrolling fashion to the user.
ISYS_MSG_Titles	"S"	Establish titles. The remainder of the block is an ASCIIz string which serves as a title to the streaming messages that follow.
ISYS_MSG_Rewrite	"R"	Rewrite Streaming Message. Similar to "M" except the ASCIIz string should replace the last streaming message displayed.
ISYS_MSG_Beginscan	"A"	Indicates that during a CREATE or UPDATE, the DLL has started its scanning phase, walking the disk directories to determine what indexing is to be done.

ISYS_MSG_Scanprogress	“B”	During a scanning phase, indicates progress. The remainder of the block consists of an Smallint indicating which parameter is being updated and a Longint which contains the new value for the parameter. Use the constants provided in the header files to decode the parameter constants.
ISYS_MSG_Deindexing	“C”	Indicates that the de-indexing phase of an UPDATE has commenced.
ISYS_MSG_Deleting	“D”	Indicates progress during the deletion phase. The remainder of the block contains a Longint representing the number of words deleted so far and an ASCIIz descriptive string.
ISYS_MSG_Reading	“E”	Indicates the start of the “reading documents” phase of an UPDATE and precedes the arrival of message type “F”.
ISYS_MSG_Readprogress1	“F”	Indicates progress in reading documents. The remainder of the block contains three longints representing the number of documents indexed, words processed and documents remaining; two 20-byte ASCIIz strings representing the elapsed time and disk space remaining, and a Longint representing the number of file-open failures.
ISYS_MSG_Readprogress2	“H”	Has the same layout as message type “F”, and indicates the index is being updated. The longints represent the percentage of the index update which is complete, the number of index references added so far, and the total number to be added. The two strings represent the elapsed time and the total number of words in the index. The final longint represents the number of new words never before seen.
ISYS_MSG_Updating	“G”	Indicates the “updating index” phase of an UPDATE has commenced and precedes the arrival of message type “H”.

**Constants for ISYS\_MSG\_Scanprogress**

<u>Constant</u>	<u>Value</u>	<u>Definition</u>
ISYS_MSG_Progress_Dirs	1	number of directories or other data sources scanned to date
ISYS_MSG_Progress_Files	2	number of operating system files or other data sources scanned to date
ISYS_MSG_Progress_Rulefiles	3	number of documents covered by rules
ISYS_MSG_Progress_Index	4	number of documents to be indexed
ISYS_MSG_Progress_Deindex	5	number of documents to be de-indexed
ISYS_MSG_Progress_Reindex	6	number of documents to be re-indexed

**6.2 IDB\_Function CREATE**

To set up an initial set of index files, or to clear out an existing index file, call IDB\_Function with “CREATE” as its parameter. ISYS loads the definitions of special characters from the file ISYS.CFG and the common-word list from the file ISYS.CWD. Any changes in these files require that the index be rebuilt.

<u>Parameters</u>	<u>Description</u>
REGARDLESS	If the index already exists, reinitialize it. (Otherwise it will not reinitialize the index.)

**6.3 IDB\_Function UPDATE**

To update a index to match changes in documents, call IDB\_Function with “UPDATE” as its parameter.

<u>Parameters</u>	<u>Description</u>
PRESORT	Sort the entries into chronological order.
PRESORT REVERSE	Sort the entries into reverse chronological order.
PRESORT NAME	Sort the entries into alphabetical order based on file name.
PRESORT PATH	Sort the entries into alphabetical order based on full path.



## 6.4 IDB\_Function OPTIMIZE

To optimize space and disk utilization, call IDB\_Function with “OPTIMIZE” as its parameter. IDB\_Function dumps the word, document, and reference information from the existing index files into a temporary set. These lists are then reordered to give the best subsequent access profile. The old index is reinitialized and the word, document, and reference information is read back into the index.

“OPTIMIZE” optimizes an index for speed and space. If an index has been subject to a great deal of maintenance or has had a large amount of deindexing performed upon it, some free space may be created within the index files (the “Stats” command shows you the amount of free space). Performing an “OPTIMIZE” reclaims this free space, as well as arranging logically contiguous sections of the index such that they are also physically contiguous.

It is a good idea to “OPTIMIZE” when your free space gets too high (although you may find that a complete “Reindex” will execute more quickly). You should also “OPTIMIZE” when an index is not going to have any further maintenance performed on it, i.e. ADD or UPDATE functions, and you want to ensure the index is optimally organized.

## 6.5 IDB\_Function REINDEX

When you need to reinitialize an index from scratch, call IDB\_Function with “REINDEX” as its parameter.

<u>Parameters</u>	<u>Description</u>
REGARDLESS	If the index already exists, reinitialize it. (Otherwise it will not reinitialize the index.)

## 6.6 IDB\_Function PREVIEW

When you wish to scan an index directory for changed files, call IDB\_Function with “PREVIEW” as its parameter.

<u>Parameters</u>	<u>Description</u>
DETAIL	The default is to produce a summary display.

## 6.7 IDB\_Function STATISTICS

When you wish to gather and report statistics for an index, call IDB\_Function with “STATISTICS” as its parameter.

<u>Parameters</u>	<u>Description</u>
(none)	

## 6.8 IDB\_Function FREQ

When you wish to list word frequency list for an index, call IDB\_Function with “FREQ” as its parameter. You can specify that the list contain only words that begin with a certain letter or letters, or that words are listed only if they appear a certain minimum number of times.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
<base>	The letters that the word must start with.
<count>	The minimum frequency.
TOP	Display only the 20 most populous words (which may be good candidates to be added to the ISYS.CWD file).
TOP <n>	Display the <n> most populous words.

## 6.9 IDB\_Function LIST

If you wish to list all the documents in an index, call IDB\_Function with “List” as its parameter.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
ALPHA	List documents in alphabetic sequence.
REVERSE	List documents in reverse chronological order. The default is in chronological order.
<Other>	If any other word is given as a parameter, it is interpreted as a filter. The document name must contain this string to be included in the list.

## 6.10 IDB\_Function COMMON

If you wish to list the common words in an index, call IDB\_Function with “COMMON” as its parameter. The common word file format is very simple: place one word on each line, in alphabetical order.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
(none)	

## 6.11 IDB\_Function VERSION

To get the complete ISYS version number, call IDB\_Function with “VERSION” as its parameter.

<u>Parameters</u>	<u>Description</u>
-------------------	--------------------

(none)	
--------	--

## 6.12 IDB\_Function USERS

To get the number of active and licensed users, call IDB\_Function with “USERS” as its parameter.

<u>Parameters</u>	<u>Description</u>
-------------------	--------------------

(none)	
--------	--

## 6.13 IDB\_Function SUPPORT

To activate any of a number of low-level integrity checks on the selected index, call IDB\_Function with ISYS\_IDB\_Support and one of the following strings:

<u>String</u>	<u>Definition</u>
“CHECK”	Check index integrity.
“STATS”	Displays index statistics.
“FORMATS”	Lists selected formats.

The header files contain constants that define these commands. Their names are always “ISYS\_IDB\_” followed by the command. For instance, ISYS\_IDB\_Check = “CHECK”.

Some of these calls are more useful than others. They are not used in the standard ISYS Windows applications; they are listed here only for informational purposes.

## 6.14 IDB\_Function ADD

To add documents to the index, call IDB\_Function with ISYS\_IDB\_Add. Documents that have been removed or changed will not be updated. This allows you to quickly add new documents to your index.

<u>Parameters</u>	<u>Description</u>
-------------------	--------------------

(none)	
--------	--

## 6.15 IDB\_Function RECATEGORYIZE

To force a complete recategorization of all the documents in the index, call IDB\_Function with ISYS\_IDB\_Recategorize. ISYS will regenerate its automatic categorization rules, and will then completely recategorize the documents based upon its newly derived rules and the ones you have explicitly stipulated in the ISYS.CAT file.

<u>Parameters</u>	<u>Description</u>
-------------------	--------------------

(none)	
--------	--

## 6.16 IDB\_Function PROCESSCACHEDDELETES

ISYS includes the ability to defer deindexing so it may be done more efficiently in batches. Subject to your Deferred\_Deleted\_Cache\_Percent setting, ISYS automatically ensures the cache of deleted documents doesn't grow too inefficient, and will automatically process the cache when appropriate. However, you may wish to exert some discretion over this process, and force the cached deletes, for example, to be processed every night, thereby reducing the likelihood of the cache being processed during peak periods.

When you call IDB\_Function with ISYS\_IDB\_ProcessCachedDeletes, any deferred deletions are immediately processed and the deferred deletion cache becomes empty.

<u>Parameters</u>	<u>Description</u>
-------------------	--------------------

(none)	
--------	--

## 6.17 IDB\_Function REFRESHSECURITYCACHE

If you have elected to have ISYS cache document security information (ACLs) into the index for rapid access across networks, ISYS will automatically refresh the ACL for a document whenever the document content changes. However, if you have made large scale changes to your security regime, you may want ISYS to completely reload the ACLs for all documents in the index. This could be achieved by performing a REINDEX, but would involve unnecessarily reindexing all the documents.

When you call IDB\_Function with ISYS\_IDB\_RefreshSecurityCache, all ACLs are reloaded into the index.

<u>Parameters</u>	<u>Description</u>
-------------------	--------------------

(none)	
--------	--

---

## 7. LOW-LEVEL INDEXING

The Low-Level Indexing API provides greater control over what documents are indexed, when they are indexed, and how they are identified. This API makes it possible to circumvent the normal ISYS directory-scan/batch-update approach to index building.

A typical application for this API would be a document-control system that adds or removes documents from its index in real time instead of during scheduled reindexing runs. Using this API provides a more immediate response and a more tightly integrated system than one which relies on regular directory scans to discover new, changed, or deleted documents. Most applications will be able to use the procedure `IDB_Function` described in the previous chapter. An alternate middle-ground indexing method is the Transactional approach, described in chapter 8.

To use the Low-Level Indexing API, you must call `Open_Database` with `Update_Mode = ISYS_OpenReadWrite`, indicating that you wish to write to the index files.

Six entry points comprise the Low-Level Indexing API:

<u>Routine</u>	<u>Description</u>
<code>Start_Concording_Run</code>	Initializes memory.
<code>Create_Database</code>	Creates a new index.
<code>Concord_From_File</code>	Indexes one file.
<code>Close_Concording_Run</code>	Completes the indexing process.
<code>DeConcord_From_File</code>	Deindexes one file.
<code>Complete_DeConcording</code>	Completes the deindexing process.
<code>Set_Concord_From_File_Option</code>	Sets indexing options.

**Sample call sequence for a program that only performs indexing:**

```
Init_Instance
Open_Database
Start_Concording_Run
Concord_From_File (repeat for each document)
```

```
Close_Concording_Run
```

```
Close_Database
```

```
Close_Instance
```

**Sample call sequence for a program that only performs deindexing:**

```
Init_Instance
```

```
Open_Database
```

```
Start_Concording_Run
```

```
DeConcord_From_File (repeat for each document)
```

```
Complete_DeConcording
```

```
Close_Concording_Run
```

```
Close_Database
```

```
Close_Instance
```

**Sample call sequence for a program that performs deindexing and indexing:**

```
Init_Instance
```

```
Open_Database
```

```
Start_Concording_Run
```

```
DeConcord_From_File (repeat for each document)
```

```
Complete_DeConcording
```

```
Concord_From_File (repeat for each document)
```

```
Close_Concording_Run
```

```
Close_Database
```

```
Close_Instance
```

Note that the above samples are more than just suggestions. Your call sequence must conform to the outlines above. If this is not possible, you may like to consider the Transactional Indexing API described in the chapter 8.

You can combine the Low-Level Indexing API with an External Access Module. Whether the documents are indexed via the normal rule-based directory scan or the direct API, using low-level indexing, has no bearing on the format of those documents.

It is possible to intermix the Low-Level Indexing API with the normal ISYS rule-based indexing in the same index, but combining the two indexing techniques is probably useful only in a small number of situations. Because the documents inserted into the index via the Low-Level Indexing API will not be matched up during the ISYS conventional document scan, they will be removed from the index during the next UPDATE run.

ISYS passes the 255-byte file “names” through to the lower level access modules which may include your own External Access Module to something other than an O/S file-system. Note that the names you pass *are* case-dependent. ISYS will not capitalize them for you.

The Low-Level Indexing API is considered advanced use of the ISYS engine. It imposes a number of usage rules which, if not followed, may lead to runtime errors or system crashes.

## 7.1 Procedure Start\_Concording\_Run

Allocates temporary files in the current directory and acquires memory workspace. You need to have called Open\_Database with Update\_Mode = ISYS\_OpenReadWrite, indicating that you wish to write to the index files.

Also note that your current directory must be changed to the directory in which the ISYS index resides before calling Start\_Concording\_Run.

<u>Parameters</u>	<u>Description</u>
EmsgBlk	Pointer to Error_Control_Block

## 7.2 Procedure Create\_Database

Create\_Database creates or re-creates an index in the current directory and loads the initial common word list. You could also create the index using an IDB\_Function CREATE, if you prefer.

<u>Parameters</u>	<u>Description</u>
EmsgBlk	Pointer to Error_Control_Block

## 7.3 Function Concord\_From\_File

Call Concord\_From\_File for each document that is to be added to the index. The document will be opened and indexed.

If the document format code specifies that it should be accessed through an External Access Module, then the same 255-byte name will be passed to ISYSEX32.DLL (or whatever named External Access Module DLL you have specified). If the document format code is anything else, ISYS will read the document directly.

ISYS gains processing efficiency by batching additions to the index file, so when a document

is closed, the words it contains might not be immediately available in the index. See procedure `Close_Concording_Run`. See the later section on efficiency.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
DocName	Pointer to an 255-byte ASCIIz string: “name” of the document. This is usually the drive, path, and file name, although it may be something else returned by an External Access Module.
TimeStamp	Longint: DOS style date stamp
FileSize	Longint: size of file in bytes
FormatCode	Smallint: specifies file’s format. See Appendix A.
EmsgBlk	Pointer to <code>Error_Control_Block</code>
<b>Returns</b>	32-bit reference to the document as it resides in the index. The reference is guaranteed unique and unchanging, unless an ISYS “OPTIMIZE” or “REINDEX” operation is performed, in which case the 32-bit references may be reallocated. Applications may choose to store this 32-bit reference within their own data files for reference purposes.

## 7.4 Procedure `Close_Concording_Run`

This ensures that all references are actually processed into the database. For most small runs, this is where all the indexing will take place. This call also frees memory and disk workspace.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
EmsgBlk	Pointer to <code>Error_Control_Block</code>



## 7.5 Function DeConcord\_From\_File

This provides a special deindexing option for those times where the original text of the document is still available. This is sometimes the case with document management systems and the like, and if the unchanged text is still available, ISYS can make great efficiency savings since it knows exactly what words to remove, in what order, etc.

If the original document image is not available, then deindexing is best achieved via Transactional Indexing mode, or by calling IDB\_Function UPDATE, and letting ISYS do a document walk to determine which documents no longer exist. ISYS can then bulk delete all the non-existent documents in the most efficient way possible.

Deindexing when the original is still available is achieved by calling DeConcord\_From\_File for each file you want removed from the index, passing the same name under which the document was indexed. ISYS will open the document and read its contents in order to determine which references are to be removed. Note that the document contents *must* be available for Deindexing to occur and must be exactly the same as when the document was originally indexed. To find the exact name of the document, you can call IDB\_Function with LIST as its parameter.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
DocName	Pointer to a 255-byte ASCIIz string: the path and file name of the document
EmsgBlk	Pointer to Error_Control_Block
<b>Returns</b>	Boolean: if the document cannot be found, the function returns false, otherwise it returns true

## 7.6 Procedure Complete\_DeConcording

After calling DeConcord\_From\_File for each file you want removed, then call Complete\_DeConcording which completes the Deindexing process. As with indexing, most of the work will be performed at this time, although it may also be done as the files are being handled one-by-one where large data volumes are involved.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
EmsgBlk	Pointer to Error_Control_Block

## 7.7 Procedure Set\_Concord\_From\_File\_Option

Specifies how ISYS is to perform the Concord\_From\_File routine.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
Opt	Smallint: bit-field options as to how the file is to be indexed
S	Pointer to Pascal string: used for the ISYS_CCFopt_dBaseID option, ignored for all other options
EmsgBlk	Pointer to Error_Control_Block

Opt is an integer. It can be any combination of:

<b><u>Constant</u></b>	<b><u>Value</u></b>	<b><u>Definition</u></b>
ISYS_CCFopt_Vent	1	Interpret Ventura Publisher control codes
ISYS_CCFopt_dBaseID	2	Identify dBASE records using the fieldname contained in S
ISYS_CCFopt_dBaseEntire	4	Index the entire dBASE record, not just MEMOs
ISYS_CCFopt_ASCII1	8	Document is hard-spaced, and each line ends with a hard return and paragraphs are indicated by double-hard-returns
ISYS_CCFopt_ASCII2	16	Document is double-spaced, and each line ends with two hard returns and paragraphs are indicated by three hard-returns
ISYS_CCFopt_WYSIWYG	32	Index through WYSIWYG, where possible
ISYS_CCFopt_Widelines	64	Index allowing lines wider than 78 characters. Lines will wrap at 160 characters if not wrapped earlier by a soft or hard return
ISYS_CCFopt_OEM	128	Characters should be assumed to be in the OEM character set, and ISYS will convert them to ANSI prior to indexing
ISYS_CCFopt_HTML	256	Documents will be transparently converted to HTML where possible, and indexed as HTML. Use of this option may require additional licensing fees.

To select more than one of the options, add them together. You cannot combine ISYS\_CCFopt\_dBaseEntire, ISYS\_CCFopt\_ASCII1, and ISYS\_CCFopt\_ASCII2.

## 7.8 Efficiency

The ISYS Engine is optimized to provide good performance even as data volumes increase, as a text retrieval system that can only provide performance on small quantities of data is of little use to anyone.

As such, its internal architecture is designed so that it benefits greatly from economies of scale. Consequently you will find overall performance much better if you do a great deal of work within each concordance run, than if you bracket each unit of work within its own concordance run.

In other words, the following type of processing is very inefficient:

**Example of very inefficient calling sequence:**

```
Init_Instance
Open_Database

Start_Concording_Run
Concord_From_File file_1
Close_Concording_Run

Start_Concording_Run
Concord_From_File file_2
Close_Concording_Run

Start_Concording_Run
Concord_From_File file_3
Close_Concording_Run

Start_Concording_Run
Concord_From_File file_4
Close_Concording_Run

Close_Database
Close_Instance
```

The more efficient way to achieve the same result is:

**Example of efficient calling sequence:**

```
Init_Instance
Open_Database

Start_Concording_Run
Concord_From_File file_1
Concord_From_File file_2
Concord_From_File file_3
Concord_From_File file_4
Close_Concording_Run
```

Close_Database Close_Instance
----------------------------------

Note that updating your index via Transactional Indexing mode or IDB\_Function mode frees you of these concerns, and ISYS optimizes the granularity itself.

Also remember that as long as you keep your file names consistent with what ISYS would see in its normal processing, you can always do an IDB\_Function REINDEX to recreate your entire index, or an IDB\_Function UPDATE if you suspect your index may have become unsynchronized with your data.

---

## 8. TRANSACTIONAL INDEXING

The primary method for ISYS updating its indexes is to do a full scan of the information space, and detect what information is new, what has changed, and what no longer exists. This is done by means of IDB\_Function “UPDATE”, and can be used with the built-in file interfaces and also External Access Modules. This approach is simple, reliable and reconstructable in the event of system failure.

The secondary method is the Low Level Indexing API, where applications can give explicit commands to index particular files at particular times. This offers the benefit of absolute control, but makes it difficult to reconstruct an index from scratch unless the data so indexed also matches the information space that would be seen by an IDB\_Function “UPDATE”, in which case reconstruction is possible via an IDB\_Function “REINDEX”. Additionally, using the Low Level Indexing API presents difficulties with changed or deleted information, as efficient immediate de-indexing is only available when the original text of the document is available. Otherwise, changed and deleted information must be picked up by means of a full information-space scan using IDB\_Function “UPDATE”.

The third method, described here, offers a transactional approach whereby the application constructs a transaction file containing various statements of fact, for example, “this document still exists”, “I know this document no longer exist”, and “here is a document, its identifier and its content”.

The ISYS Engine reads the transaction file and updates its index according to the statements of fact. This enables applications to update an ISYS index without necessarily having a complete view of the document set in its entirety at any one time.

The transactional approach offers the benefits of whatever degree of immediacy and granularity are required, immediate deletion or re-indexing of changed information, and some degree of fallback control. In the event of system failure, a backup copy of the ISYS index may be recovered and subsequent transaction files applied to “roll forward” again. It is the applications responsibility to make whatever backups of the ISYS index and transaction files it requires, and also to reset and roll-over the transaction file upon successful processing by ISYS.

If application document naming conventions are compatible, all three methods can be used together.

### 8.1 Function Process\_Transactions

This routine opens the specified ISYS index and applies the nominated transaction file. In addition to de-indexing entries specifically described as deleted, you can optionally cause documents that have not been “seen” for a certain period of time to expire and be deleted.

All deletions are processed first, then all additions. A changed document is processed as a deletion followed by an addition. It is your responsibility to ensure the transaction file makes sense – ISYS will *not* pre-scan the transaction file to identify entries which could be more efficiently expressed. For example, if a transaction file contains transactions to add a particular document, then another transaction to delete that same file, it is important to understand that the deletion will be attempted *before* the addition. The deletion will fail, and the addition will succeed. It would be far better to pass ISYS a transaction file which did not mention that document at all.

<u>Parameters</u>	<u>Description</u>
Database	Pointer to an ASCIIz string: the full path name of the ISYS index to which the transaction file is to be applied
TransFile	Pointer to an ASCIIz string: the full path and file name of the transaction file to be applied
Expire	Longint: zero to not time-out and expire documents, otherwise set this to the number of minutes beyond which, if a document has not been heard of, will be assumed to no longer exist and will be removed from the index
EmsgBlk	Pointer to Error_Control_Block

While processing is being performed, a series of informative messages will be streamed to the hWnd specified in the Init\_Instance call. See the IDB\_Function call for more information on how these messages are formatted.

Any errors are returned through the Error Control Block. The transaction file is not erased or otherwise altered by ISYS. It is the applications responsibility to inspect the EmsgBlk, and if no error was flagged, to reset the transaction file.

## 8.2 Transaction File Format

The transaction file is a binary file that must conform to a strict specification.

The general format is:

Header
Transaction
Transaction
Transaction

Each transaction type indicates a certain item of fact that ISYS will assimilate into its index.

Documents are identified by a document “name”, which is completely arbitrary but must be 255 characters or fewer in length, and must uniquely identify that document.

Document names do not necessarily have to correspond to physical file names or anything else with a definite existence. However, if they do correspond, you will be able to also perform IDB\_Function “REINDEX” to reconstruct your index without reference to any transaction files.

Further, at query time, if you attempt to open a document that was indexed from a transaction file, ISYS will attempt to open the document name in the usual way. Some applications never ask ISYS to open a document, they just use ISYS to retrieve lists of filenames. Other applications will ask for the files to be opened, in which case if the document names you are using do not correspond to physical file objects, you will want to access them via an External Access Module or a Foreign File System.

All date/times used herein are in standard four byte DOS notation, represented as two 16-bit values (least significant byte first)

<b><u>Element</u></b>	<b><u>Bits Used</u></b>	<b><u>Values</u></b>
Day	0-4	1-31
Month	5-8	1-12
Year	9-15	0-119 (year biased by 1980)
Seconds	0-4	0-29 (multiply by 2 to get seconds)
Minutes	5-10	0-60
Hours	11-15	0-24

### 8.3 Header

The transaction file header is four bytes in length and is the characters “ISTX”. This signature lets ISYS confirm that you are passing a transaction file intended for it.

The header must be present, or ISYS will reject the transaction file.

### 8.4 Confirm Document Exists (type 1)

This transaction type is used in conjunction with the Expire parameter on the call to Process\_Transactions.

Its purpose is to confirm the fact that, as at a certain time, the named document was known to still exist. ISYS updates its index to reflect the fact. If the Expire parameter is used, then documents that have not been “seen” for that number of minutes will be timed-out and removed from the index.

Use of this transaction type and the Expire parameter go hand-in-hand. That is, if you use the Expire parameter to time-out documents, you need to also use this transaction type to periodically confirm the document exists. If you do not confirm the existence of the document, then it will time-out and be de-indexed “Expire” minutes after initial creation. Similarly, there is no point in generating this transaction type unless you are using a non-zero Expire parameter in your call to Process\_Transactions.

The format of the transaction is:

Item	Length	Data Type	Contents
TransCode	1	Byte	Value 01
DocName Length	1	Byte	Length of DocName following
DocName	n	Char	Document name, not zero terminated
Time Stamp	4	Date/Time	Time document was seen to exist

The time stamp specified is typically just the current system time when the document was seen to exist. This is the time that is recorded in the ISYS index for this document, and that is used to determine when to time-out and expire the document.

If the document name specified cannot be found in the ISYS index, this transaction is ignored.

## 8.5 Document No Longer Exists (type 2)

This transaction type is used to indicate that a document no longer exists and should be removed from the index.

It can be used instead of, or in conjunction with the expiry mechanism provided by transaction type 1 and the Expiry parameter.

All transaction type 2 records in a transaction file are processed en-masse at once. There are substantial economy of scale performance benefits to be reaped by doing multiple deletions per transaction file.

In other words, one transaction file with 50 deletion records in it is much, much more efficient than 50 transaction files, each containing a single deletion. ISYS does not simply



process each deletion as it comes upon them, it processes all deletions per transaction file simultaneously.

The format of the transaction is:

Item	Length	Data Type	Contents
TransCode	1	Byte	Value 02
DocName Length	1	Byte	Length of DocName following
DocName	N	Char	Document name, not zero terminated

If the document name specified is not found in the index, this transaction is ignored.

## 8.6 New or Changed Document (type 3)

This transaction type indicates a new document exists, and specifies the content of that document. Nothing external to the transaction file is needed to load documents.

New documents benefit from economies of scale as well, so it is much more efficient to have one transaction file that adds 100 new documents, than to process 100 small transaction files, each of which adds one new document.

The format of the transaction is:

Item	Length	Data Type	Contents
TransCode	1	Byte	Value 03
DocName Length	1	Byte	Length of DocName following
DocName	n	Char	Document name, not zero terminated
Reserved	4	Longint	Zero
Time Stamp	4	Date/Time	Date and time of the document following
File Type	1	File Type	File type code (see appendix A), or 33 for Auto logic
Length	4	Longint	Length of document following
Document	n	Document	Actual document

The time stamp provided is the creation time or last modification time of the document. This is not the time stamp used for expiry purposes. ISYS automatically updates the expiry time stamp while processing the transaction. The last modification time stamp is used when sorting query results.

The document following is any type of document that ISYS can index if it were a freestanding file. For example, word processor files, spreadsheet, etc. The File Type field should be set to one of the file type constants listed in Appendix A, or to the value 33, in which case ISYS will use its AUTO logic to determine what type of file it is. If you specify a particular file type, ISYS will attempt to index the file as that type. If it cannot index it as that type, it will fail that file.

You may use this feature in conjunction with External Access Modules, as ISYS will place the document into a temporary storage area and then open that temporary file conventionally. Your EAM will just see itself being called on the temporary file, and will not know that it was actually sourced from a transaction file.

Optional indexing control can be set using the Set\_Concord\_From\_File\_Option routine.

If the file specified is new, it will be added to the index. If the file specified already exists in the index, it will be de-indexed and then re-indexed.

Note: it is crucial that you do not place duplicates in the transaction file. Duplicates will result in duplicate entries in the index, which will subsequently cause index failures when de-indexing. If your application is such that duplicates may occur in the transaction file, you should deduplicate the file before passing it to ISYS.

## 8.7 Indirect New or Changed Document (type 6)

This transaction is similar to type 3, but instead of the body of the document being supplied as a binary object within the transaction file, it is instead obtained from elsewhere. In using this transaction type, you should be sure that the document really will be available when required. The advantage of transaction type 3, by comparison, is that although the transaction files tend to be larger, they are completely self-contained.

The format of the transaction is:

Item	Length	Data Type	Contents
TransCode	1	Byte	Value 06
DocName Length	1	Byte	Length of DocName following
DocName	n	Char	Document name as it will be recorded in the index, not zero terminated
TempName Length	1	Byte	Length of TempName following, or zero if there is no TempName
TempName	n	Char	Optional: name of file from which to read the document. If no tempname is given, then the DocName itself will be opened
Time Stamp	4	Date/Time	Optional: date and time of the document. If not specified, ISYS will use the actual date/time of the DocName or TempName
File Type	1	File Type	File type code (see appendix A), or 33 for Auto logic
IndexingOptions	2	Word	Indexing options, using same coding scheme as Set_Concord_From_File_option
bDelete	1	Boolean	Indicates whether TempName should be deleted after the document is processed

Note: for SQL, file type must be SQL (not AUTO or ASCII), and the TimeStamp must be zero.

## 8.8 Rename Document (type 7)

This transaction is useful when you have renamed a document, and avoids having to perform a deindex/add pair of transactions. It is dramatically more efficient than the alternative.

The format of the transaction is:

Item	Length	Data Type	Contents
TransCode	1	Byte	Value 07
OldName Length	1	Byte	Length of OldName following
OldName	n	Char	Document name as it is currently known in the index, not zero terminated
NewName Length	1	Byte	Length of NewName following
NewName	n	Char	Document name as it should be renamed to

It is the application's responsibility to ensure that the “old name” exists, and to ensure that the “new name” does not exist, and to ensure that other transactions in the same transaction file do not invalidate those conditions.

## 8.9 Non-Unicode Code Page (type 8)

This transaction is useful when you are adding non-Unicode documents to a Unicode index, and the documents do not contain their own expression of code page, and by some other means, you know what code page the documents should be interpreted as.

For example, you do not need this transaction for HTML files which include a CHARSET directive, as that already tells ISYS the code page. You also do not need it for Unicode documents. You would only use it, for example, with ASCII files where you knew the code page (by some means), and it was different to the default system code page.

The format of the transaction is:

Item	Length	Data Type	Contents
TransCode	1	Byte	Value 08
Codepage	4	Longint	Non-Unicode code page

Once a code page transaction has been processed, it stays in effect for the remainder of the transaction file. It is valid to have multiple transactions of this type within the one transaction file.

---

## 9. EXTERNAL ACCESS MODULES

The ISYS DLL knows how to read a large number of file formats (See Appendix A). It is, however, the nature of the software industry that new file formats are always being devised. If you have a file in a format that ISYS cannot read, you may write a file access module that will enable ISYS to read the file. The file access module you write is a DLL. ISYS will call your DLL to scan, index, and retrieve documents.

The ISYS DLL knows how to read files from any operating system volume mounted on the computer where it is running. You may want ISYS to read documents that reside on some other file system. The External Access Module API allows you to write code that will enable ISYS to read files from such an external file system.

You may have a generic DLL (called ISYSEX32.DLL) which applies to all your ISYS indexes, or specifically named DLLs which are specifically loaded via your ISYS.CFG file on a per-index basis.

Once your DLL is installed, the ISYS front ends behave exactly as they did before, except that a new class of documents, as defined by your access module, are now visible to ISYS. You can write your own ISYS front-end to use in conjunction with your External Access Module, even though the two do not directly interact. Finally, you can use the ISYS DLL to find the documents for you and then open them directly in your front-end.

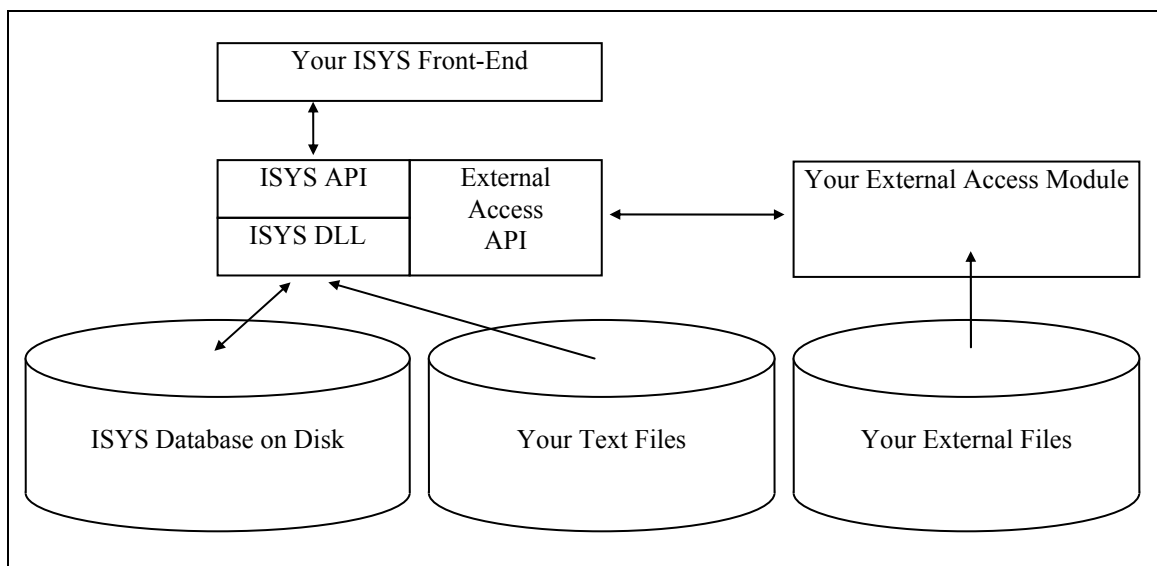


Figure 2. An External Access module gives ISYS the ability to read external file formats.

You may write your External Access Module DLL in any language that can create a Microsoft Windows DLL. Sample code in Pascal and C is provided on the ISYS SDK disk. You should name your DLL "ISYSEX32.DLL" for generic EAMs which apply system-wide, or any other name for EAMs which will only be loaded on a per-index basis. Be aware that 32-bit DLL entry-point names are case sensitive!

ISYS will automatically look for and use the DLL, if it finds it. ISYS searches, in order, the following directories:

- the current directory,
- the Windows directory,
- the Windows System directory,
- the directory with the executable file,
- the directories listed in the PATH environment variable, and
- any directories mapped in your network.

A single EAM DLL may contain multiple External Access Modules or even act as a switch between alternate access module DLLs. However, usually loading separate DLLs via your ISYS.CFG is the best approach.

There are two basic parts to the External Access Module system described in this chapter. One part deals with files and their names. It is discussed in "Determining What Gets Indexed." The other part deals with the contents of files. It is discussed in "Accessing the Documents."

## 9.1 Script EAM Alternative

Note that from version 7 onwards, ISYS provides the ability to achieve much of what External Access Modules can achieve through simple scripting. Before going down the EAM path, you may like to review the scripting capability as documented in the ISYS Utilities Help file.

## 9.2 Naming and Loading your EAM

If you name your EAM ISYSEX32.DLL it becomes a "global" EAM and is potentially available to all ISYS indexes. Actual usage of the EAM for each index depends on the index including the keyword "EXTERNAL" in its "FORMATS" list. The EAM is loaded according to standard Windows LoadLibrary() behavior, and must be placed such that LoadLibrary() will locate it. In other words, in your executable directory, current directory or on the PATH.

Alternatively, you may give your EAM a custom name and explicitly load it on an index by index basis. In this situation, use the "LOAD" keyword in your ISYS.CFG file to specify which EAM applies to that index. You must still use "EXTERNAL" in your "FORMATS".



For example:

**Example ISYS.CFG with a named EAM:**

```
LOAD MYEAM.DLL

FORMATS EXTERNAL
```

The “LOAD” directive may either give a fully specified path to your nominated EAM DLL, or just the DLL name itself, in which case the DLL must be locatable according to standard LoadLibrary() conventions.

You may only have one EAM loaded per index. If you need to have the functionality of multiple EAMs at the same time, then either break your index into separate indexes with one EAM per index and chain them together for retrieval, or simply create one “super” EAM which includes the multiple functionality you require.

## 9.3 Determining What Gets Indexed

The External Access Module system gives you the ability to specify your own method of traversing a directory of text files.

The files section of the ISYS External Access Module system is modeled on document files residing in directories on a hard disk. You may, however, write an External Access Module that reads any file that contains text. You could, for example, read text blobs in relational databases: when ISYS references directories, your module references database files; when ISYS wants to open an individual file, your module gives it a database record.

During an UPDATE run, ISYS scans the disk directory to see what files exist and determines which should be indexed, de-indexed, re-indexed, etc. The External Access Module system provides the same functionality through two calls, Ext\_Scan\_First\_Directory and Ext\_Scan\_Next\_Directory. ISYS calls these routines repeatedly until they return a value that means no more entries exist. The calls must return these parameters:

- Name, a 255-byte document name or unique identifier that ISYS can later quote back to the interface to read the data that was just scanned.
- Stamp, a 32-bit generation counter or a time stamp that ISYS will use to determine whether documents have changed since indexing. (Like the DOS date-time stamp)
- Size, a 32-bit integer containing size in bytes of the document file or blob.

You must arrange necessary configuration files to control what databases or directories the interface scans when ISYS calls the scanning routines.

An External Access Module can also be written to let ISYS access standard documents on a

foreign file system, one with names that do not follow DOS conventions. To provide this ability, your EAM must provide two additional calls, `Ext_Request_Document` and `Ext_Release_Document`. You must also call `Ext_Foreign_File_System` to turn it on or off.

## 9.4 Accessing the Documents

This section describes how ISYS reads individual documents through your External Access Module. ISYS will call `Ext_Open_Document`, quoting the 255-byte document “name” that was previously returned during a scan. Your module opens the document and returns a reference to it. If the document cannot be opened, your module returns an error code. ISYS may ask the interface to open several documents at once.

Your module may impose some limit on the number of documents that can be opened at once, since ISYS (and Windows) have such limits themselves. For example, an appropriate maximum might be 32. If this limit is exceeded, the access module could display an error message box and return an error code.

Once a document is open, ISYS may call `Ext_Get_Time_Stamp`, quoting the reference returned in the open call. Your module should pass back the current date/time or generation stamp. ISYS may also call `Ext_Get_File_Size`, where the interface should pass back the size of the document in bytes.

ISYS may call `Ext_Read_Character`, quoting the document reference and a 12-byte “context” area. Your module should return the next sequential character from the file and update the context area with positional information that will enable it to re-seek to the same character when necessary. When the file is initially opened, the 12-byte area will contain all zeros.

ISYS may call `Ext_Seek`, quoting a 12-byte context that your module should re-seek back to. Note that ISYS may make these calls in any sequence, but the interface may be sure that the context area will only be loaded with values that the interface itself had passed back to ISYS previously. After seeking, ISYS may then continue to call `Ext_Read_Character`.

Because ISYS may wish to start reading from any point within the document, your module should not keep any context information anywhere except in the 12-byte context. This restriction is relaxed to the extent that your External Access Module can derive contextual information from any arbitrary location in the file. Note however, that it is completely acceptable for the interface to keep contextual information between calls to `Ext_Scan_First_Directory` and `Ext_Scan_Next_Directory`, as these occur in a single threaded manner and are completely sequential.

## 9.5 External Indexing

Some applications, for example document management systems, may need to store document-level application-defined data within the ISYS database to have this information available when a query is performed.

The ISYS API provides entry points for doing this, `Set_Indexing_Callback_Hook` and `Set_Indexing_Filter_Hook`. The application-defined information is stored in the `ISYS.IXC`

file in the space normally used by the document title. By default, the first line of the document is assumed to be the title. This feature generally makes sense in the context of an External Access Module. If you use this facility of the API, then you cannot use the normal ISYS “document title” feature.

## 9.6 External Access Module API Routines

The External Access Module DLL which you write must define and export certain routines. You must give your routines certain names; the parameters for your routines must precisely follow the definitions given in this section. You cannot write an EAM in any language that does not allow you to create DLLs. You do not need to write your own front-end to the ISYS DLL to use your custom EAM. That is, the regular retail version of ISYS:desktop with its predefined user interface will make use of your EAM automatically and transparently.

- To write an External Access Module that lets standard ISYS read another file format, you must supply File-Access routines. Edit the ISYS.CFG file by adding a line that defines which file types should be read through your EAM. For example, the added line here will cause ISYS to give files with the file type .EXT in or under C:\DIR\ to the External Access Module, and process everything else via an AUTO rule:

### Sample Code:

```
UNDER C:\
    EXTERNAL      DIR\**\*.EXT      add this line
    AUTO          DIR\**\*.*
```

## File-Access Routines

<u>API Routine</u>	<u>Description</u>
Ext_Open_Document	Open a document so ISYS can index or display it.
Ext_Get_Time_Stamp	Tell ISYS when the document was last modified.
Ext_Get_File_Size	Tell ISYS how many characters the file contains.
Ext_Read_Character	Return the next character from the file.
Ext_Seek	Go to a specific character in the file.
Ext_Close_Document	Close the file.

If the “documents” in your data are not in discrete files — as in a relational database that contains text — then write an External Access Module that contains File-Directory routines and File-Access routines. Edit the ISYS.CFG file manually by adding the word “EXTERNAL” to the end of the Formats line, so it looks like this:

FORMATS ASCII SOURCE RTF WINWORD SPREADSHEET EXTERNAL

“EXTERNAL” *may* be the only other word in the Formats line. You usually would not add an “EXTERNAL \*.XYZ” to the rules area of the ISYS.CFG file.

## File-Directory Routines

<u>API Routine</u>	<u>Description</u>
Ext_Scan_First_Directory	Return name of first file eligible for indexing.
Ext_Scan_Next_Directory	Return name of subsequent files eligible for indexing.

To write an External Access Module that lets ISYS access a foreign file system while using ISYS’ own file readers, you must supply File-Directory and External-Filesystem Routines and call Ext\_Foreign\_File\_System. A foreign file system is one in which the files are not accessible via normal DOS file primitives and must be “requested” from some other controlling entity.

## External-Filesystem Routines

<u>API Routine</u>	<u>Description</u>
Ext_Request_Document	ISYS requests a file from an external file system.
Ext_Release_Document	ISYS releases a file back to the external file system.
Ext_Foreign_File_System	Tell the ISYS Engine whether to use external file system.

If you wish ISYS to access documents that are in a format ISYS does not know, live in groupings other than in separate files, and are available only through the mediation of special filesystem access methods, then you will need to supply routines from File-Access, File-Directory, and External-Filesystem groups.

The following routines provide additional functionality to your External Access Module:

## Additional External Access Module Routines

<u>API Routine</u>	<u>Description</u>
Set_Indexing_Callback_Hook	Tell ISYS how to call your indexing routine.
<i>MyIndexingCallback</i>	Your indexing routine.
Set_Indexing_Filter_Hook	Tell ISYS how to call your filtering routine.
<i>MyFilteringCallback</i>	Your filtering routine.
Ext_Prepare_For_WEP	Prepare for unloading.
Ext_Get_Names	Allows EAMs format names to be shown in the retail product of ISYS.
Ext_Select_Output_Coding	Allows your EAM to be Unicode aware.

## 9.7 Function Ext\_Scan\_First\_Directory

## 9.8 Function Ext\_Scan\_Next\_Directory

When ISYS indexes, it must first make a list of all eligible files. It does this by calling this pair of routines. First it calls `Ext_Scan_First_Directory` once, then it calls `Ext_Scan_Next_Directory` repeatedly. `Ext_Scan_First_Directory` must set up whatever is necessary and return the name of the first file. ISYS will continue calling `Ext_Scan_Next_Directory` until it returns a 0, meaning that there are no more files.

<u>Parameters</u>	<u>Description</u>
Name	Pointer to a Pascal string: an arbitrary 255-byte identifier, generally a DOS filename. It does not actually have to correspond to anything in the disk directory or mean anything to ISYS; it only has to be meaningful to your External Access Module.
DateTime	Pointer to a Long: gives the date stamp of the file
Size	Pointer to a Long: gives the length of the file
<b>Returns</b>	Smallint: -1 = OK, 0 = none found. Your procedure generates this value and returns it to ISYS.

These two routines vaguely correspond to the DOS functions `FindFirst` and `FindNext`. The names of these routines are misleading because they actually return the names of individual files, not directories. If your External Access Module needs to access a database of records instead of a directory of files, then return a unique identifier for each eligible record. For example, use the record number converted to a string. If you want your External Access Module to scan more than one disk directory, you have to write the code to make it do that.

It does not matter exactly what you return as the file name as long as when you get that same string back from ISYS in another call, you can refer back to the same document or record.

Ext\_Scan\_First\_Directory must locate the directory entry for the first document and return its unique identifier (name), date/time stamp, and size in bytes. Ext\_Scan\_Next\_Directory must locate the directory entry for the next document and return the same information about it. Documents do not have to be returned in the same order each time they are scanned.

If you want to rely on ISYS' built-in directory scanning code, then you do not have to provide implementations for Ext\_Scan\_First/Next\_Directory in your External Access Module. You must still export the routine names, but each routine can simply always return zero.

## 9.9 Function Ext\_Request\_Document

ISYS calls this routine to request a file from your external file system. This routine is to fetch the requested file, make it available as a DOS file, and return the DOS file's name.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
Name	Pascal string containing the file identifier that your External Access Module returned in an earlier ISYS call to Ext_Get_First/Next_Directory.
<b>Returns</b>	Pascal string containing the DOS file system path and name of the file. This string must contain a filename in standard DOS filename syntax; the ISYS file-type detector depends on that syntax to determine file types. If the call must fail, return an empty string.

If you want ISYS to call this routine, then supply it as part of your External Access Module and call Ext\_Foreign\_File\_System or enable foreign file system processing via your ISYS.CFG.

## 9.10 Procedure Ext\_Release\_Document

ISYS makes this call when it has finished accessing the file. Your External Access Module can then do whatever it needs to with the file retrieved for the Ext\_Request\_Document call.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
Name	Pascal string: contains the same string as the External Access Module returned to ISYS in the Ext_Request_Document call

If you want ISYS to call Ext\_Release\_Document, then supply it as part of your External Access Module and call Ext\_Foreign\_File\_System.

## 9.11 Procedure Ext\_Foreign\_File\_System

Your front-end to ISYS may call this procedure to turn foreign-file access on and off. Use this call if you want to have ISYS call Ext\_Request\_Document and Ext\_Release\_Document.

<u>Parameters</u>	<u>Description</u>
ForeignFilesOn	Boolean word:  True: foreign-file access is on and ISYS will attempt to call Ext_Request_Document and Ext_Release_Document.  False: foreign-file-access is off . ISYS will not attempt these calls.

Do not turn the foreign file system on or off while ISYS has files open. Do not turn foreign-file access on unless you also provide an External Access Module with the necessary calls.

Ext\_Foreign\_File\_System is not a part of the DLL you write. ISYS provides this call in its API. It is documented here because it is associated with the External Access Module system.

Note that you can also activate Foreign file system logic via your ISYS.CFG file. Just place the appropriate keyword in your index configuration file. This saves you from making the API call.

### Example ISYS.CFG with Foreign file system handling enabled:

```
ForeignFileSystem
```

## 9.12 Function Ext\_Open\_Document

Once a list of files to be indexed has been generated, ISYS will request your External Access Module to open your files. ISYS will give the name of the file you must open. If you returned names in Ext\_Scan\_First/Next\_Directory, the name will be one of those. Your External Access Module may have to open several documents at the same time.

When you open the file, you must create a unique identifier for that file and return it in the hDoc parameter. Whenever ISYS wants to refer to that particular file, it will specify it by using the value you return.

<u>Parameters</u>	<u>Description</u>
Name	Pointer to a Pascal string: contains the document name
Returns	Smallint: hDoc, document reference. Your procedure generates this value and returns it to ISYS. If there was an error and your procedure could not open the file, return -1 for hDoc.

## 9.13 Function Ext\_Get\_Time\_Stamp

ISYS uses the time stamp associated with a file to determine whether the index is up-to-date. This routine is how ISYS gets the time stamp for files associated with your External Access Module. If your “files” are records in a database, then you might want to add a field that remembers when each record was updated and returns its value for this call. Otherwise, the entire file must be updated when a single record changes.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
hDoc	Word: document reference, the same as returned by Ext_Open_Document
<b>Returns</b>	Longint: date/time stamp or generation counter. Your procedure generates this value and returns it to ISYS.

The value returned by this procedure has no specific meaning to ISYS except that if the value is *different* than when the file was first indexed, it must be reindexed. The value can actually contain anything: a date, a generation counter, checksum, etc.

If you use the ISYS DLL to locate the files for indexing, the DateTime parameter is most likely the DOS date-time field for that file.

## 9.14 Function Ext\_Get\_File\_Size

ISYS needs to know how many characters any given file contains. If your “files” are records in a database, return the length of the part of the record that ISYS will index.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
hDoc	Word: document reference, the same as returned by Ext_Open_Document
<b>Returns</b>	Longint: document size in bytes. Your procedure generates this value and returns it to ISYS.

ISYS uses the document size reported by Ext\_Get\_File\_Size as part of its error-checking. If the file size reported by this routine is wrong, ISYS may assume that the file has been corrupted and not index it.



## 9.15 Function Ext\_Read\_Character

Ext\_Read\_Character is the central routine in your External Access Module. It is how ISYS finds out what is inside your file. Ext\_Read\_Character returns streams of characters, one byte at a time. ISYS takes care of assembling the characters into words, paragraphs, and documents.

<u>Parameters</u>	<u>Description</u>
hDoc	Word: document reference, the same as returned by Ext_Open_Document
Context	Pointer to a 12-byte context area: ISYS supplies the buffer, you fill it with the context value
<b>Returns</b>	Word: Ext_Read_Character generates this value and returns it to ISYS. The character must be in the least-significant byte.

Given an hDoc to an open document and a current context, this routine returns a byte which contains the next sequential character and updates the context. If your file contains special formatting that you want ISYS to retain, you can use these byte codes:

<u>Code</u>	<u>Definition</u>
0	Ignore this character
1	Soft space: Used by some word processors to indicate removable blank spaces used for full justification.
2	Start normal style
3	Start bold style
4	Start italic style
5	Start underline style
7	Start treating all words as common
8	Indent
9	Tab
12	Page Break
13	Soft Return: Used to indicate where a line breaks even though it is inside a paragraph. Does not cause a paragraph break in ISYS indexing scheme.
14	Hard return: Indicates a new paragraph.
15	Start of metadata
16	End of metadata
26	End of File

Setting a style implies that any previous style is no longer in force. You cannot assign more than one style to any given section of text.

Any other characters in the range 17-255 may be returned and will be treated as valid characters in the ANSI character set. You must use code 26 to mark the end of a document. You must use code 14 to mark the end of a paragraph. It's appropriate to use codes 12 and 13 to mark page boundaries and increase the readability of your text. Other codes are purely optional. Code zero is provided for your convenience—ISYS simply ignores it.

EAMs which need to process multibyte Asian text should simply return each half of the MBCS character in turn. In other words, it would take two calls to `Ext_Read_Character` to return a single whole Asian character. The characters should be returned in the appropriate encoding scheme for the country involved, and the `ISYS.CFG` file should be configured for character interpretation for that country.

If your underlying data is in Unicode, then you may normalize it to multibyte ANSI before passing the text to ISYS. Use the Windows `WideCharToMultiByte` API, and return each of the (possible) multibyte characters one at a time.

If you really do require full Unicode output from your EAM, then use the Script EAM facility instead, as documented in the ISYS Utilities Help file, or use the `Ext_Select_Output_Coding` entry point to make your EAM Unicode-aware.

Character codes 15 and 16 are used to identify the portion of the document which contains metadata. The metadata region must appear before any non-metadata, and must be contiguous.

Character code 7 effectively allows EAMs to return a continuous flow of text while controlling which bits are indexed and which bits are not. From the appearance of a character code 7, all subsequent words are treated as though they were “common” words (for example, “it”, “a”, “the”). By treating them as common, the words are not indexed, but meaningful proximity searching can still be done. The attribute stays in effect until a character code 2 is returned to revert to normalcy. Note that it is essential that your EAM makes consistent and reproducible decisions of where to generate the 7 codes and the 2 codes, even following an `Ext_Seek`.

If you need to use other attributes for whatever purposes you require, simply use any byte code not mentioned above that is not in the displayable range you need to use. ISYS will treat those characters as any others, and will pass them through and return them when you browse the document using `Get_Document_Line`. The meaning of these codes can be whatever you desire – ISYS will not attempt to interpret them in any way, other than the normal significant / insignificant / delimiter interpretation configured in the `ISYS.CFG` file.

When you return a character, ISYS may want to be able to go back to that exact character. Along with each character, you return a context: 12 bytes whose format you define. The context can contain anything you wish, as long as each character returns a unique context and you can find any character by its context. In a simple ASCII file, the context is best used as a simple byte offset from the beginning of the file. Other file formats are more complicated, so you can use as much or as little of the 12-byte area as you need. Please also refer to `Ext_Seek`.

`Ext_Read_Character` is called more often than all the others combined. We recommend that

you spend most of your efforts at optimization here. The speed of this routine affects two things: indexing speed and document browsing speed. Since ISYS uses the indexes to satisfy queries, query speed is not affected.

## 9.16 Procedure Ext\_Seek

When ISYS builds an index, it stores words and the contexts where they can be found. When the front-end program asks for a particular hit within a file, ISYS translates that into a context. Your External Access Module has to translate that into a character in the file.

When ISYS calls Ext\_Seek, it is asking your External Access Module to start reading at the character specified by the context. The next call will be Ext\_Read\_Character. The character read by that call must be the same character that was originally read and returned with the context that was quoted in the Ext\_Seek.

<u>Parameters</u>	<u>Description</u>
hDoc	Word: document reference
context	Pointer to 12-byte context area

## 9.17 Procedure Ext\_Close\_Document

When ISYS is finished with a given document, it calls Ext\_Close\_Document. Your External Access Module should then perform the steps necessary to close the file associated with the document reference number: close files, release memory, etc.

<u>Parameters</u>	<u>Description</u>
hDoc	Word: document reference

## 9.18 Ext\_Initialize

This is an optional call that your EAM can export. ISYS calls this immediately after loading your DLL but before calling any other entrypoints.

Your EAM does not have to export this entry point. ISYS will link to it only if it is defined.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
hLib	Longint: the module handle of the ISYS8.DLL which caused your DLL to be loaded
hISYS	Smallint: a handle to the ISYS instance which caused your DLL to be loaded

## 9.19 Ext\_Finalize

This is an optional call that your EAM can export. ISYS calls this when the DLL is about to be unloaded from memory. It is similar in function to Ext\_Prepere\_For\_WEP, except it is supplied with additional information and is called before Ext\_Prepere\_For\_WEP.

Your EAM does not have to export this entry point. ISYS will link to it only if it is defined.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
hISYS	Smallint: a handle to the ISYS instance which caused your DLL to be loaded

## 9.20 Ext\_Prepere\_For\_WEP

This is an optional call that your EAM can export. ISYS calls this before unloading the EAM DLL, which is where we recommend you perform any closing of databases, etc. This is because Windows unloads cascaded DLLs in opposite sequences depending on whether they are statically or dynamically linked, and in your normal WEP, you may find yourself wanting to call lower down DLLs that have already been unloaded. When your Ext\_Prepere\_For\_WEP is called, the whole unload process has not yet started, so you can be sure your DLL is still in an environment in which it can call other DLLs, regardless of whether it has statically or dynamically loaded them.

Your EAM does not have to export this entry point. ISYS will link to it only if it is defined.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
(none)	

## 9.21 Procedure Ext\_Get\_Names

This procedure allows the programmer of the EAM to “name” their format for use with the UI of the retail version of ISYS:desktop. If this procedure is present, it will retrieve the name given in the parameters of this procedure for use in the UI. If this procedure is not present, but you still are using an EAM, it will put “External OEM DLL” as the format in the UI.

Note that your SDK license does *not* permit you to redistribute ISYS:desktop. You may integrate your EAM with ISYS:desktop for your own testing purposes, to create a product which licensed ISYS:desktop customers are able to make use of, or for use with a redistributed ISYS:desktop subject to specific licensing for this purpose.

<u>Parameters</u>	<u>Description</u>
LongName	Pascal type string: 32 characters for display in listboxes
ShortName	Pascal type string: 11 characters which appears next to the filename as indexing is running or in the “IDB LIST” output

## 9.22 Procedure Ext\_Auto\_Format

This procedure allows the programmer of the EAM to become involved in ISYS’ automatic file format detection.

The function, if exported, is passed a filename (which may be the name of a temporary file), the extension of the original document, a pointer to first kilobyte of file, and AUTO's estimation of the file format so far. The extension is the actual file extension, which may be different to the filename (for example, in the case of an FTP or ZIP tempfile; filename will be XYZ0123.TMP, extension might be DOC). The format passed is what AUTO is going to return if left to its own devices.

The EAM may choose to set Format any way it likes, for example, to override something AUTO has selected and make it BYPASS, or to return a type of External if it wants the file to be processed by the EAM.

<u>Parameters</u>	<u>Description</u>
DocName	Pascal type string: the name of the document, which may be the name of a temporary file.
Extension	Pascal style string: the extension of the original document
Format	Pointer to a longint: the file format as ascertained by ISYS so far (as enumerated in Appendix A). The EAM may set this to a new value.
pFirstK	A pointer to the first kilobyte of data from the file.

## 9.23 Ext\_Select\_Output\_Coding

Under normal circumstances, EAMs should always return text encoded in ANSI, regardless of whether the ISYS index is being normalized into ANSI or Unicode. ISYS will internally convert the characters if required.

EAMs also have the option of being “Unicode-aware EAMs”. This class of EAM is obligated to return ANSI characters for ANSI indexes, and Unicode characters for Unicode indexes. The engine informs the EAM how it should behave.

To become a Unicode-aware EAM, export an entry point with the name “Ext\_Select\_Output\_Coding”. If you do not export this entry point, then you are not Unicode-aware and should always return your characters in ANSI.

ISYS will call your entry point immediately after calling Ext\_Open\_Document, indicating whether your EAM should return its text in Unicode or in some other nominated code page. Unicode text is encoded in UTF-8 representation.

<u>Parameters</u>	<u>Description</u>
hDoc	Word: document reference
Code	Longint: contains the value CP_UTF8 if the EAM is required to return text in the Unicode character set, or otherwise a code page identifier if the text is to be returned in ANSI.

Note that use of this entry point is optional.

## 9.24 EAMs and IFilters

Note that if you wish to index documents for which an IFilter is available, you do not have to write an EAM. You can simply include an explicit indexing rule in your ISYS.CFG, for example:

```
IFILTER *.XYZ
```

---

## 10. INSTALLABLE SECURITY FILTERS

At indexing time, ISYS respects normal operating system security when performing disk and volume scanning. When indexing via a transactional mode or Low Level Indexing API, the programmer controls the enumeration of the name space, and thus has complete control over which document objects are seen by the engine and which are not.

By default, at retrieval time ISYS builds its result list based on all the documents it finds in the index that match the query. It returns these documents in the result list without considering whether or not the documents are allowed to be accessed in the context of the current user. Thus, by default, a query shows all matching documents, even those that may not ultimately be readable. If a user attempt to open a document to which access is not allowed, the open fails.

Additionally, you may opt to Enforce Visible Security via your ISYS.DEF configuration file. In this case, each item on the result list is checked for accessibility before being included in the list.

This can have a substantial impact on performance, but has the benefit of ensuring that only authorized documents are mentioned in the result set. The performance impact can be substantially ameliorated through the use of `Cached Security` in the `CFG_Settings` structure.

The ISYS engine automatically respects the operating system security settings for disk files. For other types of data, such as SQL data or custom OEM data, you may not want to take advantage of this security filtering feature, but insert your own filter.

This is done by creating a DLL called `ISYSSEC.DLL` and placing it in the same directory as the `ISYSAuth.dll`. When you deploy your application, you will also need to deploy `ISYSAuth.dll`.

<b><u>API Routine</u></b>	<b><u>Description</u></b>
Auth_Initialize	Optionally implement to execute initialization code.
Auth_Uninitialize	Optionally implement to execute cleanup or termination code
Auth_Activate	Activate the user context based on AuthString
Auth_Deactivate	Deactivate the currently selected user context
Auth_Openable	Returns if the specified document is to be added to the result list
Auth_GetLastError	Return extended information about the last error that occurred
Auth_GetSystemName	Return the system name used to identify the security system.

## 10.1 Auth\_Initialize

ISYS calls this routine at least once before making any other calls. This routine may be called many times, and your code will need to handle this possibility. It may also occur that your DLL is unloaded and reloaded in some occasions - a call to Auth\_Uninitialize will occur before the file is unloaded.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
None	

## 10.2 Auth\_Uninitialize

ISYS calls this routine just before it unloads your DLL. Any clean-up or termination code should be executed here.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
None	

## 10.3 Auth\_Activate

ISYS calls this routine to change user context. This method gives you the opportunity to validate user credentials.



<b><u>Parameters</u></b>	<b><u>Description</u></b>
lpAuthString	Pointer to an ASCIIz string that specifies the authentication string of the new user.
<b>Returns</b>	If the function succeeds, the return value is true.

## 10.4 Auth\_Deactivate

ISYS calls this routine to revert user context.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
<b>Returns</b>	If the function succeeds, the return value is true.

## 10.5 Auth\_Openable

ISYS calls this routine for each document it adds to the result list. If the function indicates the document is not authorized, then the result list is built as though the document had not been found by the query.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
--------------------------	---------------------------

lpIndexPath	Pointer to an ASCIIz string that specifies the path to the index to which this document applies.
lpFileName	Pointer to an ASCIIz string that specifies the document name, as record in the index, which ISYS would like to add to the result list.
Reserved1	Longint: reserved
Reserved2	Longint: reserved
<b>Returns</b>	Longint: return code, as below.

ISYS_Auth_Unknown	-1	The document was unexpected or can't be authenticated.
ISYS_Auth_OK	0	The document has been authenticated and can be added to the result list.
ISYS_Auth_NotFound	1	The specified document could not be located.
ISYS_Auth_NoRights	2	The user is not authorized to view this document. This may result in the user being challenged for new user credentials.

## 10.6 Auth\_GetLastError

ISYS calls this routine to fetch extended information about the last error that has occurred.

<u>Parameters</u>	<u>Description</u>
lpBuffer	Pointer to a buffer to receive the ASCIIz string containing the error message.
lpSize	Pointer to a longint which contains the maximum size, in <b>CHARs</b> , of the buffer specified by the <i>lpBuffer</i> parameter. On return, populate with the size of the return string
<b>Returns</b>	If the function succeeds, the return value is true.

## 10.7 Auth\_GetSystemName

ISYS calls this routine to ascertain the system name that should be used to identify this system. The name returned from here is used in ISYS AuthStrings to identify this system.

<u>Parameters</u>	<u>Description</u>
lpBuffer	Pointer to a buffer to receive the ASCIIz string containing the system name.
lpSize	Pointer to a longint which contains the maximum size, in <b>CHARs</b> , of the buffer specified by the <i>lpBuffer</i> parameter. On return, populate with the size of the return string

## 10.8 ISYS Auth Strings

ISYS passes user context information to your security filter by the means of an ISYS Auth String. The ISYS Auth String contains all the information required to activate a user, in a simple name\value pair, semi-colon delimited list, as shown below:

```
SYSTEM=NT;USER=guest;PASSWORD=guest;DOMAIN=localdomain
```

ISYS:web will automatically change the USER and PASSWORD properties of the Auth String to match a given user, leaving the unknown properties. This allows you, as the developer, to add extra information in the Auth String independent of ISYS.

## 10.9 User Contexts

Security Filters are designed to support two types of operating environments - Single User (such as ISYS:desktop) and Multi User (such as ISYS:web).

In a Single User environment, there is generally only one user context; the logged on user. This user does not change during the course of executing the program. In such an environment, you cannot rely on Auth\_Activate or Auth\_Deactivate on being called, but still must be able to respond to Auth\_Openable.

In a Multi User environment, there is generally more than one user active on the system. Due to this, each user must be activated before any security can be evaluated. In such an environment, each call to Auth\_Openable will be contained within calls to Auth\_Activate and Auth\_Deactivate.

## 10.10 Using Security Filters with ISYS:web

Before you begin using your Security Filter with ISYS:web, you must configure ISYS:web with an Anonymous Auth String. The Anonymous Auth String is used to identify users who have not yet authenticated against the server, this user usually has the lowest level of security available. ISYS:web will then use this Anonymous Auth String as a template for all other Auth Strings.

---

# 11. ADVANCED INTEGRATION

This chapter covers functions that allow you to integrate with ISYS on a lower level. When using these functions, the programmer must be very careful since the possibility of index corruption is present.

## 11.1 Integration Routines

<u>API Routine</u>	<u>Description</u>
Direct_IXC_Read	Get a file's indexing header.
Direct_IXC_Write	Set a file's indexing header.
Lookup_Document_By_Name	Returns IXC number for an indexed document name.
Rename_Indexed_Document	Changes the name of a document in the index.
Get_Document_Id	Efficiently access document unique identifier.
Get_Chained_Database_List	Gets information about a chained index.
ISYS_Multiplex	Utility features that can be performed.
Switch_Instance	Alternate between multiple Engine instances.
Set_Indexing_Callback_Hook	Obtain control as each document is indexed.
Set_Indexing_Filter_Hook	Impose your own indexing namespace filter.

## 11.2 Function Direct\_IXC\_Read

## 11.3 Function Direct\_IXC\_Write

You can manipulate the IXC record after the index has been indexed. These two routines accept an indexed document file path (as would be shown in an IDB\_Function LIST operation) and read or write the associated IXC document control record. The ISYS index must be open before calling either of these routines. In the case of the write routine, the index must be opened in update mode. If not, the function will return a value of -2.

This routine should be used with caution on chained indexes because the document name may not be guaranteed to be unique across chains.

<b>Parameters</b>	<b>Description</b>
FilePath	Pointer to ASCIIz string: contains path to indexed document file
DCR	Pointer to Chapter_Entry_Block
EmsgBlk	Pointer to Error_Control_Block
<b>Returns</b>	Smallint: status of operation, zero if successful

If successful, returns zero. If the FileName could not be found, -1 is returned. It may be beneficial to perform an IDB\_Function LIST to see exactly what filenames are indexed.

### **Chapter\_Entry\_Block**

FilePath	Pascal String of up to 255 characters: full file name.
Title	Pascal String of up to 150 characters: contains the title of the file. If you set the title, you must ensure the length byte is set correctly.
Time_Stamp	Longint: date code for when the file was last modified
FileType	Byte: code for the file type (see the section on Low-Level API Calls)
Word_Count	Longint: number of words in the document
Indexed_Words	Longint: number of words from the document that appear in the index
Option_String	Pascal String [20]: internal
Reserved	15 bytes
Line_Count	Longint (valid for non-WYSIWYG only)
File_Size	Longint
Indexed_Stamp	Longint: date/time code that file was added to index
Text_Date	Longint: first date appearing in body of document
Confirm_Date	Longint: date/time file was last confirmed as being up to date
Non_Unicode_Codepage	Longint: for non-Unicode documents stored in Unicode indexes, this indicates what codepage the document should be read in.
Reserved	8 bytes
Metadata_Lines	Smallint: number of lines of metadata at the top of the document
ACL_Block	Longint: block number of cached ACL information
Para_Limit_Exceeded	Boolean byte: did document contain more than 65535 paragraphs
Metadata_Stamp	Longint: checksum of injected metadata
Entity_Block	Longint: block number of entities detected
Entity_Count	Longint: number of entities
Para_Count	Word: number of paragraphs
Flags	Byte: associated document existence bitmap
Reserved	24 bytes

## 11.4 Function Lookup\_Document\_By\_Name

You can find the IXC record number for a document, taking in account for chained indexes, by using this procedure.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
FileName	Pointer to ASCIIz string containing the name of the file to find
EmsgBlk	Pointer to error message control block
<b>Returns</b>	Longint: the IXC number for the document, 0 if FileName not located

The FileName parameter must match exactly to how it appears to the ISYS index. To check how the filename is stored in the index, you can call the IDB\_Function with LIST as the parameter.

## 11.5 Function Rename\_Indexed\_Document

This function allows you to change the name of the file that is stored in the ISYS index. It does not change the name of the file on disk.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
OldFilename	Pointer to an ASCIIz string that is the filename to change from
NewFilename	Pointer to an ASCIIz string that is the filename to change to
EmsgBlk	Pointer to error message control block
<b>Returns</b>	Smallint: result of the operation

Compare the return with the following constants:

<b><u>Constant</u></b>	<b><u>Value</u></b>	<b><u>Definition</u></b>
ISYS_Rename_OK	0	The function completed successfully.
ISYS_Rename_OFN	1	The OldFilename does not exist in the index.
ISYS_Rename_NFN	2	The NewFilename is already present in the index.
ISYS_Rename_Mode	3	The index was not open in update mode.

The OldFilename parameter must match exactly to how it appears to the ISYS index. To check how the filename is stored in the index, you can call the IDB\_Function with LIST as the parameter.

## 11.6 Function Get\_Document\_Id

Some applications find it useful to access the internal “unique” number ISYS allocates to each indexed document. The number is “unique” per index, and unvarying until a REINDEX or OPTIMIZE is performed.

This call provides a simple and efficient way of transforming a number in the range 1 to QRB.Num\_Docs into the internal document number.

In the case of a chain of indexes all being searched at once, the document number returned is the document number within a particular chain element, and the DbPath parameter is set to indicate which item in the chain the document was found in.

<b>Parameters</b>	<b>Description</b>
DocNum	Longint: document number in the range 1 to QRB.Num_Docs
DbPath	Pointer to a caller-supplied buffer that will be filled with the ASCIIz representation of the fully qualified path to the actual index in which the document was found.
DbDocNum	Pointer to a longint which will be set to the internal ISYS document number; effectively the documents IXC record number.
EmsgBlk	Pointer to Error_Control_Block

## 11.7 Procedure Get\_Chained\_Database\_List

For a chained index, this procedure returns the number of indexes in the chain and fills a caller-supplied data structure with a copy of the ISYS chained index structure.

<b>Parameter</b>	<b>Description</b>
ChainedDB	Pointer to an array, Concat_Block, that will be filled by the DLL.
NumConCat	Pointer to Smallint: set this to how many entries your ChainedDB parameter can hold, and after the call it will return set to the number of entries actually used.
EmsgBlk	Pointer to error message control block

### **Concat\_Block**

An array of elements, each which the following structure:



Dir	Pascal string [255]
Num_Docs	Longint: number of documents in the index
Reserved	Longint
Reserved	Longint
Reserved	Longint
Reserved	Longint

## 11.8 Procedure ISYS\_Multiplex

This procedure is used to supply some added functionality to the ISYS API and the way the indexes will act. Depending on the function value, the procedure will act on it as described below. It is used as a general purpose extension mechanism to avoid disturbing the published ISYS API when adding additional specific functionality.

<u>Parameters</u>	<u>Description</u>
Func	Smallint: the function to perform as listed below
Param1	Pointer to a Longint used for some functions
Param2	Pointer to a Longint used for some functions
EmsgBlk	Pointer to error message control block

The current functions available for this procedure are:

**Constant:** ISYS\_Multi\_Thresh **Value:** 1

**Definition:** Sets the threshold of occurrences above which a word will be treated as common during retrieval to the value passed in as Param1.

**Constant:** ISYS\_Multi\_Ver **Value:** 2

**Definition:** Tells ISYS not to check for index version signatures if Param1 is passed as non-zero. Turning this feature off may corrupt your index, and is strongly discouraged.

**Constant:** ISYS\_Multi\_Line

**Value:** 3

**Definition:** With the parameter Param1 = 0, returns the current maximum line width for the currently opened document in the Param1 parameter.

**Constant:** ISYS\_Multi\_SetMsg

**Value:** 4

**Definition:** Determines what Windows message number is used when the ISYS Engine sends a notification message to a nominated hWnd. The default is \$100. Call with Param1 equal to the desired message number for the ISYS Engine to use from that point onwards.

**Constant:** ISYS\_Multi\_TimeStamp

**Value:** 5

**Definition:** Retrieves the indexed date/time stamp for a found document without actually opening the document. Call with Param1 set to the document number in the current retrieve list, and Param2 will be set to the time stamp recorded in the index for that document.

**Constant:** ISYS\_Multi\_DBRoot

**Value:** 6

**Definition:** Call with Param1 pointing to a caller-supplied area of memory, and the ISYS Engine will fill that memory with the ASCIIz representation of the currently open ISYS index. In the case of a chain of open indexes, this will return the directory name of the index in the chain that was most recently active. For example, if you call Get\_Document\_Record and then make this call, you can determine which index in the chain the found record belonged to (although Get\_Document\_Id is a better method of this).

**Constant:** ISYS\_Multi\_AuditID

**Value:** 8

**Definition:** This call controls what user identifying information the ISYS Auditor feature will record in the audit log files. Call with Param1 pointing to one ASCIIz identifying string, and Param2 pointing to another. Neither parameter may be a null pointer, but either may point to an empty string.

**Constant:** ISYS\_Multi\_Msgbox

**Value:** 10

**Definition:** Controls whether ISYS Engine will display application modal messageboxes when very serious critical errors occur. By default, critical errors are displayed. Call with Param1 zero to disable messageboxes, or non-zero to enable.

**Constant:** ISYS\_Multi\_Conflate**Value:** 12

**Definition:** Controls whether conflation is performed widely, using many tense forms of the word, or narrowly, using just plurals and past tense. Default is broad. Call with Param1 non-zero to switch to narrow conflation, or zero to revert to broad conflation.

**Constant:** ISYS\_Multi\_PassThruTabs**Value:** 14

**Definition:** Normally, ISYS takes any TAB characters (code 9) in the input stream and converts them to soft spaces for efficient processing in draft mode. If you make this call with Param1 non zero, then the TAB characters are left undisturbed.

**Constant:** ISYS\_Multi\_Direct\_IXC**Value:** 15

**Definition:** Provides the ability to directly read IXC index records. Param1 is the document unique identifier and param2 points to a caller-supplied area of memory which will be filled with a Chapter\_Entry\_Block. This is an efficient way of accessing all document details if you know the internal document number.

**Constant:** ISYS\_Multi\_Docname\_to\_IXCNumber**Value:** 16

**Definition:** Param1 points to an ASCIIz document name, and Param2 will be set to the IXC internal document unique identifier, or zero if the document cannot be found in the current index. The document name must be given exactly as it is recorded in the index.

**Constant:** ISYS\_Multi\_IXCNumber\_to\_HitNumber**Value:** 17

**Definition:** Param1 is an IXC internal document unique identifier. ISYS examines the result list, and returns in Param2 a number in the range 1 to QRB.Total\_Hits indicating where in the rlist hits for that document commence. If the document is not in the result set, then Param2 returns zero.

**Constant:** ISYS\_Multi\_IXCNumber\_to\_DocNumber**Value:** 18

**Definition:** Similar to the preceding call, but returns a number in the range 1 to QRB.Num\_Docs. The resulting number could be passed to Open\_Document, if desired.

**Constant:** ISYS\_Multi\_EAM\_Sense\_IXCnumber **Value:** 19

**Definition:** Some External Access Modules (EAMs) have need to know what internal document unique identifier ISYS is going to assign to the document they are generating. EAMs may make this multiplex call to the engine *after* Ext\_Open\_Document has been called, and Param1 will be set to the document number that will be assigned.

**Constant:** ISYS\_Multi\_Get\_Index\_Stats **Value:** 20

**Definition:** Param1 returns the total number of documents in the currently opened index, and Param2 returns the number of indexed words (not counting common words).

**Constant:** ISYS\_Multi\_Define\_Syn\_Char **Value:** 21

**Definition:** Param1 points to a character that ISYS will use in place of the “+” character for synonym handling in queries.

**Constant:** ISYS\_Multi\_DocNumber\_to\_Name **Value:** 23

**Definition:** Param1 is in the range 1 to QRB.Num\_Hits, and Param2 points to a caller-supplied buffer which is filled with 255-byte Pascal string of the document name as held in the index. This is a much more efficient means of determining document names than Get\_Document\_Record, as it does not cache, and performs a partial read of the IXC.

**Constant:** ISYS\_Multi\_Get\_Language\_Code **Value:** 24

**Definition:** Param1 will be set to the language code for the currently open index.

**Constant:** ISYS\_Multi\_Set\_Indexing\_Date\_Limit **Value:** 25

**Definition:** Call this with Param1 set to a DOS format date/time. Next time an IDB\_Function is performed, only files with a date/time stamp later than this date/time will be considered in the name space enumeration. This allows you to impose a time curfew from which older documents will not be included in the index.

**Constant:** ISYS\_Multi\_Get\_Document\_Access\_Method **Value:** 28

**Definition:** Call this with Param1 set to point to an ASCIIz document name. Upon return,

Param2 will be as a bitmap indicating any special access method requirements of the document:

1	Document resides in a ZIP file
2	Document resides in a CHM file
4	Document resides in a binder
8	Document resides in an email
16	Document is an attachment to an MSG file
32	Document resides in a DMS
64	Document is an attachment to an EML file
128	Document is an attachment to a Lotus Notes record

;

**Constant:** ISYS\_Multi\_Set\_Unfound\_Open\_File\_Type      **Value:** 31

**Definition:** Normally when you call Open\_Unfound\_Document, ISYS performs automatic file format detection on the document. To over-ride this, call this multiplex with Param1 set to a format code as enumerated in Appendix A. This multiplex operates as a one-shot, and after the next Open\_Unfound\_Document, the behavior will revert.

**Constant:** ISYS\_Multi\_Get\_Rlist\_Extent\_For\_Doc      **Value:** 37

**Definition:** Call this with Param1 set in the range 1 to QRB.Num\_Docs, and it will return with Param1 and Param2 set to the first and last Rlist entry numbers for that document, as may be used in a Get\_Hit\_Pointer call.

**Constant:** ISYS\_Multi\_Get\_Current\_Instance\_Handle      **Value:** 40

**Definition:** Returns with Param1 set to the current ISYS instance handle, same as would have been returned in your Init\_Instance call.

**Constant:** ISYS\_Multi\_Get\_Current\_Document\_Names      **Value:** 41

**Definition:** Call this to determine the name of the currently open document. Both Param1 and Param2 must be set to point to a caller-supplied 256 byte buffer. Upon return, the buffer pointed to by Param2 will be set to the document name as recorded in the index, and the buffer pointed to by Param1 will be filled with either the same name, or a temp file name, if the document is not already directly openable. It is valid to pass zero for either

parameter.

**Constant:** ISYS\_Multi\_Soft\_Fail\_Chain\_Opens

**Value:** 42

**Definition:** When you open a chain of indexes, ISYS verifies that every index in the chain opens correctly, and if any of them fail, then the entire chained open fails. If you call this multiplex with Param1 non-zero, then any failures within subsequent chain opens are simply ignored, and the remainder of the chain is processed regardless. This should be used with caution.

**Constant:** ISYS\_Multi\_Set\_Nonunicode\_Codepage

**Value:** 44

**Definition:** To alter the non-Unicode codepage setting, call this multiplex with Param1 set to the codepage to which you wish to switch. This may be useful, for example, when using Concord\_From\_File. The non-Unicode codepage setting is used when processing non-Unicode documents into a Unicode index, where the document format cannot express its own codepage (ASCII files, for example), and where you know the codepage of the document, and where it is different to the default system codepage.

**Constant:** ISYS\_Multi\_Disable\_Term\_Coloring

**Value:** 48

**Definition:** ISYS includes individual identification of search terms, such that they may be individually rendered in different colors. When Get\_Document\_Line is used, every start of hit marker (character code 6) is followed by a term identifying character. If you call this multiplex with Param1 non-zero, then this behavior is disabled and Get\_Document\_Line performs as in version 6.

**Constant:** ISYS\_Multi\_Line\_Number\_to\_Paragraph\_Number

**Value:** 49

**Definition:** Call this with Param1 set to the handle of an open non-WYSIWYG document, and Param2 set to a line number, and this function will return with Param2 set to the paragraph number in which that line occurs. Paragraph numbers 1 to 255 are reserved for metadata, and the body of documents are numbered from 256 onwards.

**Constant:** ISYS\_Multi\_Get\_Paragraph\_Extent

**Value:** 50

**Definition:** Call this with Param1 set to the handle of an open non-WYSIWYG document, and Param2 set to a paragraph number, and the function will return with Param1 and Param2 set to the first and last line numbers of that paragraph. Paragraph numbers 1 to 255

are reserved for metadata, and the body of documents are numbered from 256 onwards.

**Constant:** ISYS\_Multi\_Get\_Deferred\_Deletion\_Status

**Value:** 55

**Definition:** Call this with an index currently open, and it will return with Param1 set to the number of documents currently in the deferred deletion cache, and Param2 set to the number of indexed words in those documents.

**Constant:** ISYS\_Multi\_Get\_Index\_Status

**Value:** 56

**Definition:** Call this with an index currently open, and it will return with Param1 set to the total number of words in all the documents in the index, and Param2 set to the date/time stamp of when the index was last updated.

**Constant:** ISYS\_Multi\_Get\_Term\_Existence\_Map

**Value:** 61

**Definition:** Get\_Term\_Id\_Term maps a term identifier into an actual search term, however not all search terms may appear in all documents, for example if you search for “cat or dog”, some documents may contain only the first, only the second, or both. Call this multiplex with Param1 set to the handle of an open document, and it will return with Param2 set to an existential bitmap of which terms actually occur in that document. The least significant bit corresponds to the lowest Term\_Id, and so on.

**Constant:** ISYS\_Multi\_Set\_Fatal\_Error\_Log\_Dir

**Value:** 62

**Definition:** When the ISYS engine encounters a fatal error, it journals a message to a log file called ISYSERROR.LOG which is placed in the executable directory. To over-ride placement of this file, call this multiplex with Param1 set to an ASCIIz string representing the directory in which to place the file. Note that you should specify only the directory name, not a file name as well. The directory must exist.

**Constant:** ISYS\_Multi\_Get\_Index\_Created\_Software\_Ver

**Value:** 64

**Definition:** Call this with an index currently open and it will return indicating what version of ISYS8.DLL was used to most recently CREATE or REINDEX that index. The high order word of Param1 contains the major version number, and the low order word the minor version number. The high order word of Param2 contains the release number, and the low order the build number.

**Constant:** ISYS\_Multi\_Get\_Index\_Indexed\_Software\_Ver      **Value:** 65

**Definition:** Similar to the ISYS\_Multi\_Get\_Index\_Created\_Software\_Ver multiplex, except it returns indicating what version of ISYS8.DLL was most recently used to update the index.

**Constant:** ISYS\_Multi\_Disable\_Auto\_Index\_Backups      **Value:** 66

**Definition:** When called with Param1 non-zero, this multiplex suppresses automatic index backups, if they have been configured via the ISYS.CFG file. To re-enable index backups, call with Param1 equal to zero. Note that doing so will not cause automatic backups to occur for indexed which do not have the appropriate settings in their ISYS.CFG – the purpose of this multiplex is to control whether such settings are respected or not.

**Constant:** ISYS\_Multi\_Force\_Auto\_Index\_Backup      **Value:** 67

**Definition:** If an index is configured for automatic index backups to be taken, this multiplex causes a backup to be taken. Normally, backups are taken every time the index is opened with write intentions, but this multiplex allows you to initiate a discretionary backup. The index must be closed at the time of the call, and the call may take some time to complete, depending on the size of the index.

**Constant:** ISYS\_Multi\_Get\_Rich\_HTML\_Real\_DocType      **Value:** 101

**Definition:** This call is of use where documents have been indexed with Rich HTML. When a found document has a file type of HTML, make this call with Param1 set to a number in the range 1 to QRB.Num\_Docs, and Param2 will be set to the real document type of the underlying document before it was dynamically converted to an HTML view.

**Constant:** ISYS\_Multi\_DocNumber\_to\_associated\_image      **Value:** 103

**Definition:** Make this call with Param1 in the range 1 to QRB.Num\_Docs, and Param2 pointing at a caller-supplied buffer to be set to an ASCIIz filename of any image associated with that document. This is useful if you want to know that a document has an associated image, but do not want the substantial overhead of calling Get\_Annotation\_Index\_Page.



**Constant:** ISYS\_Multi\_Add\_Hits\_to\_HTML**Value:** 105

**Definition:** This function can only be called with an HTML Raw Codes document currently open. Param1 points to a caller supplied area of memory which will be filled with an ASCIIz name of a temporary file into which will have been created a copy of the HTML document, but with hit highlights added and hit to hit navigation inserted. All URLs will have been adjusted to compensate for the new location of the document. It is the callers responsibility to delete the temporary file when it is no longer needed.

**Constant:** ISYS\_Multi\_Set\_OpenUnfound\_Doc\_attributes**Value:** 107

**Definition:** When you call Open\_Unfound\_Document, ISYS first attempts to lookup the document name in the index. If the document is found, it is opened using the document options (WYSIWYG, width, etc) as recorded in the index. If the document is not found, this lets you specify what options to apply when the document is opened. The multiplex is as a one-shot causing the next Open\_Unfound\_Document to use the options specified. Param1 holds the options, using the same codes as Set\_Concord\_from\_file\_option.

**Constant:** ISYS\_Multi\_Set\_Agent\_Root\_Directory**Value:** 109

**Definition:** The Intelligent Agent API normally manages its own storage space for a single user under the Application Data or Windows directory. This function lets you redirect that storage elsewhere, either for your own purposes, or to support multiple users.

**Constant:** ISYS\_Multi\_Set\_Indexing\_Granularity**Value:** 110

**Definition:** ISYS normally manages resource consumption itself during index updates, attempting to do things in as large batches as possible. You may impose your own granularity which forces ISYS to flush its indexing run every certain number of documents. Call with Param1 set to the number of documents desired, or zero to allow ISYS to manage its own granularity. Imposing an artificial granularity will likely have a dramatic negative effect on performance.

**Constant:** ISYS\_Multi\_Set\_Instance\_Storage**Value:** 112

**Definition:** Some applications may find it useful to store data alongside each ISYS instance. Make this call with Param1 set to a single 32-bit value you wish to store in the

current ISYS instance. This value may be a pointer.

**Constant:** ISYS\_Multi\_Get\_Instance\_Storage **Value:** 113

**Definition:** Returns in Param1 the value stored via the previous multiplex.

**Constant:** ISYS\_Multi\_Get\_Document\_Changed\_Status **Value:** 115

**Definition:** Lets you explicitly check to see whether a document has been changed since indexing. With a document currently open, make this call and Param1 will be set to zero if the document has not been changed, or non-zero if the document has changed and cannot be viewed with hit highlights. Param2 must hold the handle to the open ISYS document.

**Constant:** ISYS\_Multi\_Parse\_Date **Value:** 116

**Definition:** Call with Param1 pointing to an ASCIIz string. If the string can be intelligently interpreted as a date, Param2 will be set to the DOS date/time representation of that date, otherwise to zero.

**Constant:** ISYS\_Multi\_HTTP\_Get **Value:** 118

**Definition:** Call this multiplex with Param1 pointing to an ASCIIz string which contains a URL, and Param2 pointing to a caller-supplied area of memory at least 256 bytes in length. ISYS will make a call to ISYSDPR.DLL to fetch the specified URL, and if successful, place the result in a temporary file and copy the name of the temporary file into the buffer pointed to by Param2.

**Constant:** ISYS\_Multi\_Extract\_Text\_From\_Document **Value:** 119

**Definition:** This multiplex is used to extract plain text from arbitrary document formats. Call with Param1 pointing to an ASCIIz file name, and Param2 set to a format code (as listed in Appendix A), or AUTO. ISYS will place the text content of the document into a temporary file, and alter the ASCIIz string pointed to by Param1 to become the temp file name. You must be sure that the buffer pointed to by Param1 is large enough to hold a temp file name. Should the conversion fail, Param1 will point to a null file name.

## 11.9 Procedure Switch\_Instance

As well as supporting multiple programs or multiple instances of the one program calling the ISYS Engine, ISYS also supports multiple execution contexts per program. In other words, it is valid to call `Init_Instance` several times from a single program.

The `Init_Instance` routine returns an “Instance Handle” By default, all ISYS routines work on the current instance. To switch from one instance to another (ie, one execution context to another), call `Switch_Instance()`, passing the handle of the instance you wish to use.

**For example (in pseudocode)**

```
var handle1, handle2: integer;

init_instance handle1

open_database "contacts"
perform_query "smith"

init_instance handle2

open_database "countries"
perform_query "rivers"

switch_instance handle1
get_document_record 1, mydoc

switch_instance handle2
get_document_record 1, myotherdoc

close_instance

Switch_instance handle1
close_instance
```

The API does not require an instance handle to be passed on each and every call to reduce code changes in applications using the existing API.

Note that ISYS does not support asynchronous processing of calls.

The easiest way to implement this in most situations is simply to always call `Switch_Instance(MyInstanceHandle)` upon entry to routines that make use of ISYS API calls, if your application is going to allow multiple execution instances.

Note that calls to `Switch_Instance` are very efficient, and you do not need to try to minimize use of this call.

Within linear non-multithreaded code, you can be sure that an instance, once switched in, stays switched in. You only need to worry about uncontrolled instance switching if your code is multi-threaded (not multi-tasking -this presents no complications), and operates in a preemptive multithreading environment. In this case, you should be sure you switch to the appropriate instance before each API call, and to encase each switch\_instance and subsequent API call within a critical region.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
InstHandle	Smallint: the instance to receive subsequent API calls
EmsgBlk	Pointer to error message control block

Note that version 8 of the ISYS Engine under Windows is *not* threadsafe. In situations where multithreading is required, use the COM interface to the ISYS engine which runs out of process, and therefore allows multiple threads in a single program to make ISYS calls without synchronization issues.

## 11.10 Procedure Set\_Indexing\_Callback\_Hook

Your EAM can define its own document titles separately from the normal ISYS title rules. If you want to define your own titles for documents, then use this procedure. If you do not wish to set your own document titles, then you do not need to provide this routine.

Your application should call Set\_Indexing\_Callback\_Hook before calling the IDB\_Function with the "UPDATE" parameter or before starting your second-level indexing API or Transactional processing run. When ISYS indexes files, it has to know the title that is associated with each file. This is accomplished by having ISYS call a procedure within your DLL that knows these titles. In order for ISYS to call this procedure, you must tell ISYS where it is. Set\_Indexing\_Callback\_Hook enables the application to declare a callback procedure that the ISYS engine will call when a file is being indexed.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
Callback	Pointer to your callback routine: called <i>MyIndexingCallback</i> in this documentation. You can call it what you will. Your procedure generates this value and returns it to ISYS.
EmsgBlk	Pointer to Error_Control_Block

The callback routine remains set and will be used by ISYS for the remaining life of that instance. To remove your callback hook, call the routine again passing an address of NIL (zero).

## 11.11 Procedure *MyIndexingCallback*

If you are setting your own document titles, then ISYS will call this routine to find out the document titles when it actually indexes your documents. If you do not wish to set your own document titles, then you do not need to provide this routine.

<u>Parameters</u>	<u>Description</u>
Action	Smallint: always 1
FileName	Pointer to a Pascal string: contains the name of the file being indexed. In the case of the second-level indexing API or Transactional indexing, it will contain whatever file identifier your application has chosen.
FileTitle	Pointer to a Pascal string: MyIndexingCallback can set this to any appropriate value up to 150 characters long

It is your DLL's responsibility to correctly set the length byte of FileTitle. The data portion of the string may contain any binary value, including NULL. The maximum length of the string you may set is 150 bytes (excluding the length byte). Setting a string longer than this will result in program errors.

## 11.12 Procedure *Set\_Indexing\_Filter\_Hook*

Your EAM can define its own document filter separately from the normal ISYS file filter. If you want to define your own filters for documents, then use this procedure. Your application should call *Set\_Indexing\_Filter\_Hook* before calling *IDB\_Function* with the "UPDATE" parameter or before starting your second-level indexing API run. This is accomplished by having ISYS call a procedure within your DLL. In order for ISYS to call this procedure, you must tell ISYS where it is. *Set\_Indexing\_Filter\_Hook* enables the application to declare a callback procedure that the ISYS engine will call when a file is being indexed.

<u>Parameters</u>	<u>Description</u>
Callback	Pointer to your callback routine: called <i>MyFilteringCallback</i> in this documentation. You can call it what you will. Your procedure generates this value and returns it to ISYS.
EmsgBlk	Pointer to <i>Error_Control_Block</i>

The callback routine remains set and will be used by ISYS for the remaining life of that instance. To remove your callback hook, call the routine again passing an address of NIL (zero).

If you do not wish to set your own document filters, you do not need to provide this routine.

## 11.13 Function *MyFilteringCallback*

If you are setting your own document filters, then ISYS will call this routine to find out the document titles when it actually indexes your documents.

<u>Parameters</u>	<u>Description</u>	
Code	Smallint: always 1	
FileName	Pointer to a Pascal string: the name of the file being indexed.	
<u>Returns</u>	Value	Definition
	1	ISYS will index the file according to the indexing rules.
	0	ISYS will not index the file.

If you do not wish to set your own document filters, then do not need to provide this routine.

## 11.14 Procedure *Attach\_Custom\_Relevance\_Rating*

Allows you to provide your own relevance ranking logic.

<u>Parameters</u>	<u>Description</u>
Callback	Pointer to your callback routine, called <i>MyRelevanceRanker</i> in this documentation. You can call it what you will. Your routine gets called to replace or modify the relevance rank determined by ISYS.
EmsgBlk	Pointer to Error_Control_Block

## 11.15 Function *MyRelevanceRanker*

If you have installed your own relevance rank logic, ISYS will call this routine for every document in the result list.

<u>Parameters</u>	<u>Description</u>
Document	Pointer to a Chapter_Entry_Block
Relevance	Pointer to a longint which contains the normal ISYS relevance determination for this document, in the range 1 to 100, where

higher numbers indicates greater relevance. Your DLL may alter the relevance in any way it sees fit, according to any logic you choose.

## 11.16 Procedure Merge\_Index

Merges one index into another. This may be useful, for example, in an application architecture which involves weekly indexes merging into annual indexes, of which the last five annual indexes and all the current-year weekly indexes get chained together for searching purposes. Note the CheckForDups Boolean flag – checking for and resolving duplicates takes additional time, so if you can provide an iron-clad guarantee of non-duplicate document names, you can pass zero for this flag.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
MasterIndex	Pointer to an ASCIIz string containing the full path to an ISYS index into which another index will be merged
MergeIndex	Pointer to an ASCIIz string: the index to be merged into the master
CheckForDups	Smallint: zero if you can guarantee no duplicate document names, otherwise non-zero for ISYS to first check for duplicates and ignore any merge entries which already exist in the master.
EmsgBlk	Pointer to Error_Control_Block

## 11.17 Procedure Set\_API\_Charset

Allows you to control whether strings are expressed in ANSI or Unicode character sets when passed to and from the ISYS API. Unicode strings are represented using UTF-8 encoding, and so may still be manipulated as ASCIIz strings because a single zero byte will terminate the string. Characters may occupy between one and six bytes.

Applications may switch the API from ANSI to Unicode as often as required. Indexes may be normalized to ANSI or Unicode regardless of how strings are passed back and forth through the API.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
Code	Longint: may be the Windows constant CP_UTF8 (65001) or zero for ANSI
EmsgBlk	Pointer to Error_Control_Block

## 11.18 Procedure Widechar\_to\_UTF8

Utility routine to assist Unicode UCS2 to UTF-8 string conversion. Unicode characters expressed in UCS2 encoding each occupy 2-bytes of memory, and may contain single bytes of zero within the string. Unicode characters expressed in UTF-8 encoding each occupy between one and six bytes, will never contain a single zero byte, and may hence be manipulated as normal ASCIIz strings.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
Src	Pointer to a UCS2 Unicode string, where each character occupies two bytes and the string is terminated by a double zero byte.
Dest	Pointer to a caller-supplied buffer which will hold the UTF-8 encoded equivalent. One Unicode character in Src (each of two bytes) may be converted to between one and six UTF-8 bytes.
MaxDestLen	Longint: length of the buffer pointed to by Dest (in bytes).

## 11.19 Procedure UTF8\_to\_Widechar

Utility routine to assist Unicode UTF-8 to UCS2 string conversion.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
Src	Pointer to a UTF-8 Unicode string, terminated by a zero byte.
Dest	Pointer to a caller-supplied buffer which will hold the UCS2 encoded equivalent. Each Unicode character in Src, no matter how many bytes originally used for representation, will occupy exactly two bytes in Dest. The output string will be terminated by two zero bytes.
MaxDestLen	Longint: length of the buffer pointed to by Dest (in <i>characters</i> ).



## 11.20 Procedure Get\_Document\_Security\_Descriptor

If security information is being cached in your index (as determined by the settings in Configuration\_Control\_Block), then this routine may be used to access the security information previously stored.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
DocNo	Longint: document number in the range 1 to QRB.Num_Docs
Descriptor	Pointer to a caller-supplied buffer of at least 2kb in size which will be filled with the previously cached security information
Descriptor_Size	Pointer to a longint which, upon return, will be set to the length in byte of the information copies to the Descriptor buffer
EmsgBlk	Pointer to Error_Control_Block

## 11.21 Procedure Get\_Num\_Processors

This is a utility routine which returns the number of logical and physical CPUs in a machine. In the case of hyperthreaded processors, these two numbers may be different.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
Logical	Pointer to a longint which will be filled with the number of logical processors
Physical	Pointer to a longint which will be filled with the number of physical processors
EmsgBlk	Pointer to Error_Control_Block

## 11.22 Procedure Set\_Num\_Indexing\_Processes

This controls how many parallel processors are to be used when building indexes. The default is one, or whatever is stored in your `ISYSSVR.CFG` file. In general, optimal performance will be attained when this is set equal to the number of physical processors in your machine. It is valid to set it higher, but performance may degrade.

<u>Parameters</u>	<u>Description</u>
N	Longint: the number of parallel processes to use when building indexes
EmsgBlk	Pointer to Error_Control_Block

### 11.23 Procedure Set\_Engine\_Call\_Tracing

This routine allows you to automatically log or display all calls you make to the ISYS engine. Logging may occur to a text file, a callback, or to a small pop-up window. This routine is unusual in that it is valid (indeed preferable) to call it prior to `Init Instance`.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
When	Longint: under what circumstances the tracing should occur
ISYS_Call_Tracing_Off	0 Never
ISYS_Call_Tracing_IDE_Only	1 Only when program is running with a debugger attached
ISYS_Call_Tracing_On	2 Always
Where	Longint: determines where tracing should be placed
ISYS_Call_Tracing_To_Screen	0 Display in pop up window
ISYS_Call_Tracing_To_Callback	1 Make callback
ISYS_Call_Tracing_To_Disk	2 Output to disk file

Dest	Pointer: if Where is ISYS_Call_Tracing_To_Disk, this is a pointer to an ASCIIz string containing the file name in which to place the call tracing. If this pointer is null, then call tracing will be logged to c:\ISYSEngineCalls.log. If ISYS_Call_Tracing_To_Callback, then this parameter is the address of a callback routine which accepts a single parameter in the form of a pointer to an ASCIIz string.
------	---

## 11.24 Using ISYS with other languages

ISYS can be used with most languages. The version of ISYS:desktop supplied can be taken as a good guide to how ISYS works with non-English languages.

ISYS directly supports all European languages including English. Diacritical accent marks may be either considered or ignored. If ignored, then for example all of the following characters will be considered the same for search purposes, but displayed correctly for browsing: “AaÀàÁáÂâÃã”. By default, these characters are seen as distinct (except for the upper case and lower case renditions of the same character). Treatment of accented European characters is configured through your ISYS.CFG.

Additionally, ISYS supports multibyte Asian languages. Where MBCS documents are expected (Big5, GB, Shift-JIS, etc) and the documents do not exist in a format which includes charset information, the index must be configured to tell the Engine what language conventions to expect. When running with multibyte Asian languages ensure your machine has all Asian character tables loaded.

When using ISYS with single byte languages, you should check the ISYS.CFG file for your ISYS index and ensure that your SIGNIFICANT and INSIGNIFICANT directives correctly list all the characters used by your language, both upper and lower case forms. You do not need to add Asian characters to the SIGNIFICANT character list. For Unicode indexes, you only need do this for characters with codes less than 128. Characters with codes 128 and above use operating system definitions to determine handling.

You should also review your common word settings (ISYS.CWD file), and ensure it has appropriate contents for your language.

After any changes of this type, be sure to perform a complete REINDEX of your database.

## 11.25 Combining ISYS results with other sources

It is relatively common for some SDK applications to have a need to combine the results of an ISYS query with results from some other source.

For example, in a document management application, you may have a collection of documents which have been indexed by ISYS, and a separate SQL database which contains metadata associated with each document. The document name might be some cryptic number which is never directly seen by your users, but which provides a unique key into your SQL database.

In situations like this, it is common to want to issue an ISYS query, and for each document found by ISYS, perform a lookup into the SQL database and identify the found document using the metadata, rather than a cryptic file name.

As is often the case, there are two possible approaches: an easy one, and an efficient one.

The easiest approach is for perform your query, then enumerate the result list calling `Get_Document_Record` for each found document, taking the document file name, and performing a lookup into your SQL table using the document file name as a unique key to retrieve the actual information you want to provide.

An optimization on the above approach is to call `ISYS_Multiplex` with the function code `ISYS_Multi_DocNumber_to_Name`. This is a much more efficient way to retrieve a document name than by calling `Get_Document_Record`.

The most efficient approach, however, is to not access the document names at all, since resolving a found document name takes one I/O per document. This approach makes use of the ISYS unique document identification number. This number is unvarying until an index is REINDEXed or OPTIMIZED, or a given document is replaced, and so can function as a foreign key in your SQL table.

Specifically, you would add a new column to your SQL table called `ISYS_DOC_ID`, a numeric column with an index on it. You would then populate this column by performing an “\*” query in ISYS to find all documents, and iterate through the result list calling `Get_Document_Id` to retrieve the ISYS document identifiers and `ISYS_Multi_DocNumber_to_Name` to retrieve the document file names. You would do a SQL lookup based on the file name and set the `ISYS_DOC_ID` accordingly. Within your application as new documents are created or existing documents are updated, you would arrange to maintain the `ISYS_DOC_ID` column alongside the other metadata in your SQL table. You would also want to provide some ability to re-execute the initial bulk load of the field for administration purposes.

Once this foreign key column exists in your SQL data, then dramatic performance enhancements are possible at query time. When you enumerate a result list, instead of calling `Get_Document_Record` or `ISYS_Multi_DocNumber_to_Name`, just call `Get_Document_Id` (which can be resolved with zero I/Os), and use the resulting value as a lookup into your `ISYS_DOC_ID` column.

---

## 12. RESULT-LIST MANIPULATORS

This chapter describes how to create, manipulate, and destroy lists of words found in your documents. The data structures and algorithms described in this chapter are for advanced users who wish to define their own search operators.

### 12.1 Rlist Data Structure

The locations of words are recorded in a three-part structure. The three parts correspond to the Article (or document), Paragraph, and Word. Items are listed in *reverse* order. That is, as you begin with the first item of a list and proceed to the end, the first word you encounter is the last word in the last file. You process them in reverse order until you finally arrive at the first word of the first file.

You may read rlist entries in any order, but if you create your own rlist and append entries to it, the entries must be added strictly in reverse sequence. *Unordered rlists are not permitted!*

You may, however, call `Sort_Documents`. `Sort_Documents` doesn't actually sort the rlist. When you call `Get_Document_Record`, the record returned is according to the current sort sequence, but the current\_rlist remains in its original sort order.

### 12.2 Allocation and Deallocation

When ISYS performs a search to satisfy a given command, it parses the search expression from left to right. For each term in the search expression, ISYS builds a list of hits corresponding to that term; operators between terms specify what is to be done with the lists. For example, given the search expression "cat and badger" ISYS makes two hit-lists. The first contains all the instances of the word "cat" and the second contains all the instances of the word "badger." The "and" operator traverses these lists to find documents that have words in both lists.

You must not use an rlist until you've created it. Having created it, you must subsequently deallocate it before your program ends.

### 12.3 Rlist Manipulations

To create an rlist for a specific word, you call `Perform_Find` with the word as the search string. The word would be looked up and all references to the word are listed in the "current" rlist. At this point, the "current" rlist could be copied to another rlist for later use. The time taken to evaluate a word is proportional to the size of the result.

Once you've created your rlists, either by calling `Perform_Find` or by creating them yourself and adding your own words, you can subject them to logical operations.

It would be possible to look up, for example, 100 individual words into 100 individual rlists and then to OR them all together. However, the rlist operators are typically "order N" ( $O(n)$ ) algorithms.

For example, consider the case of 100 rlists of wildly varying sizes. The first rlist contains 100,000 references and the remainder each contain a single reference. If you ORed them together in that sequence, the "order" of the evaluation would be

$$100,000 + 1$$

$$100,001 + 1$$

$$100,002 + 1$$

.

.

$$100,099 + 1 \text{ total} = 10,005,050 \text{ operations}$$

If you evaluated them in the opposite order, it would be

$$1 + 1$$

$$2 + 1$$

$$3 + 1$$

.

.

$$98 + 1$$

$$99 + 100,000 \text{ total} = 105,149 \text{ operations}$$

For synonym and wildcard evaluation, the ISYS engine performs internal optimizations. When using the rlist operators in your own application, you must calculate the optimal evaluation sequence yourself.

## 12.4 Rlist API Calls

The calls described in this chapter allow you to manipulate these lists and create your own operators.

<b><u>API Routine</u></b>	<b><u>Description</u></b>
Rlist_Create	Allocates and returns a pointer to a result-list
Rlist_Drop	Deallocates a result-list
Rlist_Get_Entry	Returns the A-P-W (Article-Paragraph-Word) for an entry in an rlist
Rlist_Append_Entry	Adds a new item to a result-list
Rlist_Get_QRB	Gets the Query Result Block—the number of entries, words, docs, etc.—for an rlist
Rlist_Empty	Empties an rlist without deallocating it
Rlist_Current_Rlist	Returns the current result rlist
Rlist_Oper	Subjects two Rlists to an ISYS operator
Rlist_Copy_From_To	Makes a copy of an Rlist
Rlist_Exchange	Exchanges the contents of two Rlists

## 12.5 Function Rlist\_Create

Allocates and returns a pointer to a new result-list.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
(none)	
<b>Returns</b>	Pointer to Result_List: the new rlist

If you allocate a result-list, then you must eventually deallocate it before your program shuts down the ISYS Engine.

**Result\_List**

A Result\_List (sometimes called rlist\_ptr) type is a pointer to the following structure:

Size	Longint: internal
Words	Word: internal
suffix	Boolean: internal
Group	Smallint: size of phrase groups
Phwrds	Smallint: number of words in phrase so far
Common	Boolean: internal
Reserved1	Longint: Reserved
Num_Arts	Longint: internal
Reserved2	String of 58 Bytes: Reserved

This structure listing is provided for informational purposes only. We recommend you do not inspect or rely upon the contents of an rlist\_ptr.

## 12.6 Procedure Rlist\_Drop

Deallocates a result-list. Once you have called this routine on an rlist-pointer, you can no longer access the list it was associated with. You can reuse the pointer by calling Rlist\_Create again.

<u>Parameters</u>	<u>Description</u>
Rlist	Pointer to Result_List: the rlist to deallocate

## 12.7 Procedure Rlist\_Get\_Entry

Returns the A-P-W for an entry in an rlist.

<u>Parameters</u>	<u>Description</u>
Rlist	Pointer to Result_List: the rlist in question
Entrynum	Longint: number of the entry in that rlist
Result	Pointer to Word_Pointer_Block: your buffer for the A-P-W entry



**Word\_Pointer\_Block**

Block that contains the A-P-W (Article-Paragraph-Word) location of a specific word in a result-list.

Word\_Num Word: contains the word number in the paragraph

Paragraph Word: contains the paragraph number in the document

Document Longint: contains the number of the document

Term\_ID Word: the id of the term the hit represents

After the call, your Word\_Pointer\_Block buffer contains an article-paragraph-word structure.

## 12.8 Procedure Rlist\_Append\_Entry

Adds a new item to a rlist.

<u>Parameters</u>	<u>Description</u>
Rlist	Pointer to Result_List: result list to append item to
Newentry	Pointer to Word_Pointer_Block: the article-paragraph-word location of the word you want to add to the rlist

## 12.9 Procedure Rlist\_Get\_QRB

Gets the Query Result Block—the number of entries, words, docs, etc.—for an rlist.

<u>Parameters</u>	<u>Description</u>
Rlist	Pointer to Result_List: the result list you want statistics for
QRB	Pointer to Query_Result_Block: a query-result block to be filled with information about the rlist

Note that Rlist\_Get\_QRB recalculates the comparative relevance scores for all the documents in the rlist. As such, it is good to make this call after you have built or modified the contents of an rlist if you subsequently plan to make use of the relevance fields.

## 12.10 Procedure Rlist\_Empty

Empties an rlist without deallocating it.

<u>Parameters</u>	<u>Description</u>
Rlist	Pointer to Result_List: result list to be emptied

After the call, the result-list, Rlist, contains no entries, but still exists. (This is not the same as deallocating a result-list.)

## 12.11 Function Rlist\_Current\_Rlist

All higher level API calls (Perform\_Find, Sort, etc.) work on ISYS' current rlist. This function allows you to access the list and make changes to it. Use this call to access the current result list.

<u>Parameters</u>	<u>Description</u>
(none)	
<b>Returns</b>	Pointer to Result_list: current result list

For instance, if you wanted to sort an rlist, the sequence would be like this:

### Code Example

```
{get a pointer to the current rlist}
Current:= Rlist_Current_Rlist;
{it may contain garbage, so empty it}
Empty_Rlist(Current);
{copy the unsorted list to the current list}
Rlist_Copy_From_To(unsorted, current);
{sort the current list--our list}
Sort_Rlist(current, sortsequence);
{retrieve elements from the sorted list in order}
For i:= 1 to CurrentSize do begin
  Get_Document_Record (i, docrec);
```

You can examine individual words or may repopulate the current result set by calling Rlist\_Empty and then Rlist\_Append\_Entry. All higher level API routines always refer to the current result set.

Do not allocate or deallocate the current result list.

## 12.12 Procedure Rlist\_Oper

Apply an ISYS operator to two result-lists.

<u>Parameters</u>	<u>Description</u>
Oper_Code	Pascal string: up to 20 characters long containing a standard ISYS operator, like “AND”, “OR”, “/10/”, etc.
RlistLeft	Pointer to Result_List: the operand-list on the left side of the operator. After the call, contains the result of the operation.
RlistRight	Pointer to Result_List: the operand-list on the right side of the operator.

For the Oper\_Code, use one of the following constants to specify the operator:

<u>Constant</u>	<u>Value</u>
ISYS_Oper_And	“AND”
ISYS_Oper_Or	“OR”
ISYS_Oper_Not	“NOT”
ISYS_Oper_Xor	“XOR”
ISYS_Oper_Except	“EXCEPT”
ISYS_Oper_ButNot	“BUTNOT”
ISYS_Oper_Label	“LABEL”
ISYS_Oper_In	“IN”
ISYS_Oper_EspIn	“ESPIN”
ISYS_Oper_Nearfollow	“..”
ISYS_Oper_Farfollow	“...”

## 12.13 Procedure Rlist\_Copy\_From\_To

If you want to make a copy of a Rlist, call Rlist\_Copy\_From\_To. This routine makes an item-by-item copy of an rlist.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
Result	Result_List: the copy of the rlist
Rhs	Result_List: the original rlist

## 12.14 Procedure Rlist\_Exchange

If you want to swap two rlists, call Rlist\_Exchange. This routine swaps the list pointers.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
Result	Result_List: one result-list
Rhs	Result_List: another result-list

---

## 13. ANNOTATIONS

The ISYS engine stores annotations in a file with the same name as the file they belong to but with a .ANN extension. The annotation files are simple ASCII and may be stored in a separate directory. ISYS never alters the original documents. The API described here provides a simple way to read and write the annotation files created by the standard retail ISYS product.

If this system does not meet your needs, then you can bypass it entirely and come up with your own. You do not have to use these routines since you can manipulate the annotation files like any other text file. They are just provided as a convenience to the programmer. However, if your intention is to read annotation files created by the retail ISYS product itself, then we recommend working through the API. The option to bypass this API and create your own annotation scheme is really only valid if you have no need to work with annotations created by other ISYS products – in other words, if the annotations are constrained to your application.

The .ANN format is fairly simple. Basically, it is an ASCII text file with lines separated by carriage-return/linefeed. The first line in the file is “ISYS Document Annotation File”. The next line begins the first annotation; subsequent annotations are separated by blank lines. Annotations can be one or more lines. The first line of an annotation contains its coordinates—the line number followed by the character number—and the type of annotation. Annotations can be of type NOTE, GRAPHIC, QUERY, DDA, or HYPER. Each type has its own parameters.

Note that the coordinate system is different between WYSIWYG and non-WYSIWYG documents. If you switch document modes, then the annotation positions will be invalidated.

### **Annotation Type    Parameter**

NOTE	Number of lines to follow
GRAPHIC	Path and file name of linked graphic image
QUERY	The query
DDA	The name of the directory to make current (optional), the name of the file to execute (on next line), and keystrokes to send to the file (optional, on 2nd line)
HYPER	The file name of the text file to link to

These can each be followed by a line that specified which icon to use to represent the link. Here is an example:

```
ISYS Document Annotation File

1 21 NOTE 1

This expedition found fossilized remains of antlers.

6 14 NOTE 1

This annotation has a special icon.

Icon ALERT.ICO

8 40 GRAPHIC c:\pictures\skull.bmp

15 24 QUERY fossil or antler

19 30 DDA c:\windows\

C:\WINDOWS\PBRUSH.EXE

airfare

24 66 HYPER records\FOSSIL.DOC 72

Icon BUTTON2.ICO
```

These routines are the calls by which ISYS allows the user to make notes about documents in a database:

<u>API Routine</u>	<u>Description</u>
Get_Annotation_Index_Page	Returns location of annotations for the document
Get_Annotation_Entry	Returns an annotation
Set_Annotation_Entry	Updates an annotation
Get_Annotation_Entry_Text	Returns text from a text annotation
Set_Annotation_Entry_Text	Updates a text annotation
Get_Annotations_Changed_Status	Reports whether annotations need to be saved
Write_Back_Annotations	Saves annotations to disk

## 13.1 Procedure Get\_Annotation\_Index\_Page

This call is the basis of the annotation mechanism and returns an index page of all the annotations that exist for the specified document. The call may be made any number of times. The document specified in each call must be open.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
hDoc	Longint: document reference returned by Open_Document
NumAnnos	Pointer to an Smallint: will be set by the DLL to indicate the number of entries in the annotation index
AnnoIx	Pointer to an array of Annotation_Entry records: buffer must be at least 10k in size
ImageName	Pointer to ASCIIz string: maximum 255 bytes. If an image is associated with the document as a whole (that is, has the same name as the document, but with an image extension, i.e. BMP, WMF, PCX ...), then the string will be filled with the file name of the associated image. Otherwise, it will be set to null.
EmsgBlk	Pointer to Error_Control_Block

### Annotation\_Entry

A call to Get\_Annotation\_Index\_Page returns a list of annotations. Each member of the list has this structure:

Row	Longint: line number of the annotation
Col	Longint: column number of the annotation
Typ	Byte: annotation type code

Annotation types are coded as follows:

<b><u>Code</u></b>	<b><u>Definition</u></b>
0	Textual note
1	Graphic Image
2	Hyperactivity
3	Query
4	Hypertext document link
5	Non specific
6	Deleted (to be ignored)

## 13.2 Procedure Get\_Annotation\_Entry

Once Get\_Annotation\_Index\_Page has been called, make this call to retrieve annotation entries in the range 1 to NumAnnos

<b>Parameters</b>	<b>Description</b>
hDoc	Longint: document reference returned by Open_Document
AnnoNo	Word: annotation number
AnnoPtr	Pointer to an Annotation_Control_Block: ISYS will fill this with the annotation
EmsgBlk	Pointer to Error_Control_Block

### Annotation\_Control\_Block

The format of this control block varies according to the type of annotation. The first nine bytes of the block duplicate the entry in the annotation index and indicate the format of the rest of the block. Because of this structure's complexity, it is defined in syntactically different but structurally equivalent ways for C, Pascal, and Visual Basic. Please see Appendix C for a full description.

Row	Longint: line number of the annotation
Col	Longint: column number of the annotation
Typ	Byte: annotation type code (see list in Get_Annotation_Index_Page)
Icon	127-character Pascal string: contains an optional name of an ICO or BMP file. This entry may be null, in which case your program should use a default icon.

The remainder of the structure depends on what it contains.

#### ***Text Annotation:***

Reserved	36 bytes. Call Get_Annotation_Entry_Text
----------	--

#### ***Graphic Annotation:***

FileNameit	255-character Pascal-style string: contains the file name
------------	---



**Hyperactivity:**

Reserved	1 byte: contains a type code
Directory	255-character Pascal string: contains DDA script
Command	255-character Pascal string: contains DDA script
Keystrokes	128-character Pascal string: contains DDA script

**Document Link:**

Doclink	255-character Pascal string: contains path and filename of document to open
---------	---

**Query:**

Query	255-character Pascal string: contains the ISYS query to execute
-------	---

### 13.3 Procedure Set\_Annotation\_Entry

This call takes the same parameters as Get\_Annotation\_Entry. It is used to maintain annotations and should be made with AnnoPtr pointing to a completed annotation block.

<u>Parameters</u>	<u>Description</u>
hDoc	Longint: document reference returned by Open_Document
AnnoNo	Word: annotation number
AnnoPtr	Pointer to an Annotation_Control_Block: contains your annotation (See Get_Annotation_Entry)
EmsgBlk	Pointer to Error_Control_Block

If AnnoNo is zero, a new annotation is created and, if it is positive, the corresponding annotation is replaced. If the annotation pointed to has its LineNo as Zero and Type as Deleted, then the corresponding annotation is deleted. If the application has added or deleted some annotations and wishes to obtain the latest copy of the annotation index, then it may call Get\_Annotation\_Index\_Page again after making a Set\_Annotation\_Entry call.

## 13.4 Procedure Get\_Annotation\_Entry\_Text

The data structure returned by `Get_Annotation_Entry` just contains a “placeholder” to the text, not the actual text itself. This means you can easily call `Get_Annotation_Entry` with only small memory overheads.

<u>Parameters</u>	<u>Description</u>
AnnoPtr	Pointer to the <code>Annotation_Control_Block</code> : contains the annotation returned by <code>Get_Annotation_Entry</code>
TextPtr	Pointer to a 32kB block: ISYS will fill this with an ASCIIz string containing the text annotation
EmsgBlk	Pointer to <code>Error_Control_Block</code>

If, having called `Get_Annotation_Entry`, you find that it is a text annotation and you want to get at the text, call `Get_Annotation_Entry_Text`. Pass in the data structure that you just got back from `Get_Annotation_Entry` and a pointer to a 32k buffer. The ISYS DLL will fill the buffer with the actual text from the annotation.

## 13.5 Procedure Set\_Annotation\_Entry\_Text

Because annotations can now be up to 32k characters long, `Set_Annotation_Entry_Text` exists as a separate call from `Set_Annotation_Entry`. To make a text annotation, first call `Set_Annotation_Entry_Text` with a pointer to the block of text. Then call `Set_Annotation_Entry` with all the usual parameters. If you are setting a text annotation, leave the text parameter empty.

<u>Parameters</u>	<u>Description</u>
AnnoPtr	Pointer to the <code>Annotation_Control_Block</code> : contains the annotation returned by <code>Get_Annotation_Entry</code>
TextPtr	Pointer to a 32kB ASCIIz string: your annotation
EmsgBlk	Pointer to <code>Error_Control_Block</code>

`Set_Annotation_Entry_Text` takes care of shifting text around inside the annotation file when you add, remove, or change the length of a text annotation.

## 13.6 Procedure Write\_Back\_Annotations

This call saves edited annotations to disk. If annotations have been maintained and the changes should be kept, call this routine. If you wish to discard the changes, then do not call this routine.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
hDoc	Longint: document reference returned by Open_Document
EmsgBlk	Pointer to Error_Control_Block

## 13.7 Function Get\_Annotations\_Changed\_Status

This call indicates whether the annotations need to be saved to disk and the name of the file they should be saved to.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
hDoc	Longint: document reference returned by Open_Document
Pstr	Pointer to an 255-byte ASCIIz buffer: will be filled by the DLL with the file name to which the annotations would be written
EmsgBlk	Pointer to Error_Control_Block
<b>Returns</b>	Smallint: non-zero if the annotations need to be saved to disk



---

## 14. ISYS CONCEPT TAXONOMIES

Users often find it quite difficult to query a database without some sort of understanding of its content. Concept Taxonomies provide a way to hierarchically organize the knowledge held in a database. For example,



This Concept Taxonomy has two main concepts, Computers and Cars. There are two concepts under Computers (Software and Hardware) and three under Cars (Sports, Off-Road, and Family). A really useful concept taxonomy would probably be much bigger than this one.

The Concept Taxonomy is created by the database administrator or publisher. Each node on the taxonomy contains a standard ISYS query; when the user selects a node, the corresponding query is executed. Nodes can also consolidate all lower nodes. That is, the concept “Cars” can be made to combine the queries associated with the three types of cars. The taxonomy is stored as a linear list. To display it as a Tree, indent each item wIndent times.

The Concept Taxonomy API provides tools for navigating and constructing Concept Taxonomies, and for executing queries from a concept taxonomy. These calls do not have to be made to modify the ISYS.SCT file. The ISYS.SCT file is just an ASCII file that can be created as a normal text file and these API calls exist as a convenience for the programmer.

<u>API Routine</u>	<u>Description</u>
SCT_Open	Opens and loads the concept taxonomy file
SCT_Entry	Returns an entry from the concept taxonomy
SCT_Replace	Replaces an entry in the concept taxonomy
SCT_Insert_After	Inserts an entry after the specified entry in the Concept taxonomy

<u>API Routine</u>	<u>Description</u>
SCT_Move	Moves or deletes an entry in the concept taxonomy
SCT_Close	Closes a concept taxonomy
SCT_Close_and_Save	Closes and saves changes to a concept taxonomy
Perform_SCT_Find	Submits a concept-based query to ISYS
SCT_Set_File_Name	Sets a non-default ISYS.SCT file name

## 14.1 Function SCT\_Open

Loads the concept taxonomy file into memory.

The default name ISYS uses for SCT\_Open is ISYS.SCT. The default name for SCT\_Close\_and\_Save is whatever it ended up using for SCT\_Open. These defaults are overridden by calling SCT\_Set\_File\_Name.

<u>Parameters</u>	<u>Description</u>
EmsgBlk	Pointer to Error_Control_Block
<b>Returns</b>	Word: number of entries in the concept taxonomy. This call may be made multiple times to ascertain the current number of entries in the taxonomy after you have inserted and deleted entries.

## 14.2 Procedure SCT\_Entry

Given a number from 1 to the number of entries in the tree, returns the SCT\_Block for that entry.

<u>Parameters</u>	<u>Description</u>
wSCTnum	Word: concept number to retrieve
SCTBlock	Pointer to an empty SCT_Block: ISYS will fill it with data
EmsgBlk	Pointer to Error_Control_Block

**SCT\_Block**

SCT_Name	ASCIIz string: length 128
Query	ASCIIz string: length 128
ORed	Word boolean: should lower levels be consolidated in queries
Indent	Word: level within the hierarchy (1 to 7)
Info	Array of 4 (array indexes 1..4 for Pascal and Basic and 0..3 for C) strings of ASCIIz string length 70: contains explanatory text
Filter	Filter_Block (see <i>Filename_Filter</i> for format)

## 14.3 Procedure SCT\_Replace

Replaces the entry for an SCT\_Block.

<u>Parameters</u>	<u>Description</u>
wSCTnum	Word: concept number to replace
SCTBlock	Pointer to new SCT_Block
EmsgBlk	Pointer to Error_Control_Block

## 14.4 Procedure SCT\_Insert\_After

Inserts the SCT\_Block supplied into the Concept Taxonomy *after* the concept number indicated.

The total number of entries in the taxonomy will be increased by one and the number for each entry following the added entry will be incremented by one.

<u>Parameters</u>	<u>Description</u>
wSCTnum	Word: concept number to insert the new entry after
SCTBlock	Pointer to new SCT_Block
EmsgBlk	Pointer to Error_Control_Block

## 14.5 Procedure SCT\_Move

Moves the indicated concept entry so that it is now in the position indicated. If wSCTto is zero, the concept is deleted and the total number of entries in the taxonomy will be decreased by one and the number for each entry following the deleted entry will be decremented by one.

<u>Parameters</u>	<u>Description</u>
wSCTfrom	Word: concept number of the entry to move
wSCTto	Word: concept number of the new position
EmsgBlk	Pointer to Error_Control_Block

## 14.6 Procedure SCT\_Close

Closes and discards the currently loaded Concept Taxonomy. This is implicitly performed when a index is closed.

<u>Parameters</u>	<u>Description</u>
EmsgBlk	Pointer to Error_Control_Block

## 14.7 Procedure SCT\_Close\_and\_Save

Closes and saves to disk the currently loaded Concept Taxonomy.

<u>Parameters</u>	<u>Description</u>
EmsgBlk	Pointer to Error_Control_Block



## 14.8 Procedure Perform\_SCT\_Find

Similar to Perform\_Find, this call retrieves the specified concept and consolidates its lower levels. Then it executes a query based on the concept. Instead of giving Perform\_Find a string containing a query, call Perform\_SCT\_Find with the number of the concept.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
wSCTnum	Word: the concept number to execute
ExtraPstr	Pointer to an ASCIIz string: will optionally be either ANDed or NOTed with the query result. This must point to a string, even if the string is empty.
wFlags	Word: quantity that specifies how the query is to be processed
hWnd	Handle: window that will receive progress messages.
QRB	Pointer to a Query_Result_Block (see <i>Perform_Find</i> for structure)
EmsgBlk	Pointer to Error_Control_Block

Use the following constants defined in the header files to set up the wFlags parameter:

<b><u>Constant</u></b>	<b><u>Value</u></b>	<b><u>Definition</u></b>
ISYS_Query_Synonym	1	synonym expansion
ISYS_Query_Conflate	2	verb conflation
ISYS_Query_Thesaurus	8	thesaurus expansion
ISYS_Query_None	0	neither conflation nor expansion
ISYS_SCTFind_And	4	“And” the query with ExtraPstr. If this flag is not specified, ExtraPstr is NOTed with the query result
ISYS_SCTFind_Not	0	“Not” the query with ExtraPstr

To use these in combination, simply add them together appropriately. However, you cannot use ISYS\_Query\_None together with ISYS\_Query\_Synonym or ISYS\_Query\_Conflate, and you cannot use ISYS\_SCTFind\_Not together with ISYS\_SCTFind\_And.

## 14.9 Procedure SCT\_Set\_File\_Name

This determines what file name ISYS will use when you next call SCT\_Open or SCT\_Close\_and\_Save. It lets you have multiple Concept Taxonomies with arbitrary names and over-ride the standard ISYS naming and location conventions.

The default name ISYS uses for SCT\_Open is ISYS.SCT. The default name for SCT\_Close\_and\_Save is whatever it ended up using for SCT\_Open. These defaults are overridden by calling SCT\_Set\_File\_Name.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
FileName	Pointer to ASCIIz string: contains the Concept Taxonomy file name
EmsgBlk	Pointer to Error_Control_Block

---

## 15. SYNONYMS

ISYS Synonyms provide the user a means to define groups of words with similar meanings. For instance, the synonyms salary, wages, and pay could be defined so that a user could enter any one of these words and have ISYS search on all three.

ISYS synonyms are stored in a simple ASCII format in the file ISYS.SYN. These synonym API routines maintain the ISYS.SYN file. Some application developers, however, may prefer to maintain their ISYS.SYN file via other means, not through this API. This is perfectly valid. The only advantage of using the ISYS API to maintain the SYN file is that changes are reflected immediately when a query is processed, whereas if the SYN file is written directly by an application, the ISYS database would have to be closed and reopened before the changes to the synonyms were reflected in subsequent queries.

Note that ISYS also ships with a predefined word thesaurus, which is not maintainable. The Synonym library, by comparison, is initially blank and may be maintained as you choose. At query time you can nominate whether synonym or thesaurus expansion takes place, or both.

<u>API Routine</u>	<u>Description</u>
SYN_Get_Head	Returns current synonym file name and pointer to the first synonym ring
SYN_Get_Entry	Returns an entry from the synonym file
SYN_Replace_Word	Replaces an entry in the synonym file
SYN_Delete_Entry	Deletes an entry from the synonym file
SYN_Add_Word_To_Ring	Adds a word to an entry in the synonym file
SYN_Undo	Discards changes made to synonym structure
SYN_Save	Saves synonym structure to disk

All synonym calls use a synonym control block, which contains the actual synonym word, plus two Longint fields that the DLL uses to represent the linkages and chains.

<b>SYN_Block</b>	
Word	ASCIIz string: 64 bytes in length
Next_Syn	Longint: internal
Next_Ring	Longint: internal

## 15.2 Function SYN\_Get\_Head

Returns the physical file name of the synonym file that is in effect and a symbolic pointer to the “head” of the first “ring” from which the whole synonym structure may be navigated.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
FilenamePstr	Pointer to an ASCIIz string: the DLL will fill this with the name of the synonym file in use
EmsgBlk	Pointer to Error_Control_Block
<b>Returns</b>	Longint: the DLL will set to the synonym head

## 15.3 Procedure SYN\_Get\_Entry

Given a reference to a synonym entry as previously provided by some other call, this routine fetches the control block associated with that entry.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
hSyn	Longint: synonym reference
SynBlock	Pointer to a client-supplied SYN_Block: the DLL will fill this
EmsgBlk	Pointer to Error_Control_Block

## 15.4 Procedure SYN\_Replace\_Word

Given a reference to a synonym entry and a string, replaces the word in the synonym block as specified.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
hSyn	Longint: synonym reference
SynonymPstr	Pointer to an ASCIIz string: up to 30 bytes long. Must be in upper case.
EmsgBlk	Pointer to Error_Control_Block

## 15.5 Procedure SYN\_Delete\_Entry

Deletes a synonym entry. The DLL rebuilds the ring around the deletion.

<u>Parameters</u>	<u>Description</u>
hSyn	Longint: synonym reference to delete
EmsgBlk	Pointer to Error_Control_Block

## 15.6 Procedure SYN\_Add\_Word\_To\_Ring

Given a word, adds the word as a synonym in the ring indicated. If the reference passed is zero, then the synonym is added as a new ring.

<u>Parameters</u>	<u>Description</u>
hSyn	Longint: synonym reference of ring to add to or zero for new ring
SynonymPstr	Pointer to an ASCIIz string: word to add. Must be in upper case.
EmsgBlk	Pointer to Error_Control_Block

## 15.7 Procedure SYN\_Undo

Discards all changes made to the synonym structure.

<u>Parameters</u>	<u>Description</u>
EmsgBlk	Pointer to Error_Control_Block

## 15.8 Procedure SYN\_Save

Saves the current synonym structure to disk.

<u>Parameters</u>	<u>Description</u>
EmsgBlk	Pointer to Error_Control_Block



---

# 16. NAMED SECTIONS

ISYS supports searching in named, multi-paragraph sections. Please see the appropriate chapter in the ISYS user manual for a description on multi-paragraph sections and labeled paragraphs. The “IN” operator is generally more applicable than named sections.

Similar to Concept Taxonomies and Synonyms, this set of routines simply maintains the ISYS.FLD ASCII file. You may, if you choose, bypass this API and maintain the file directly via your application.

<u>API Routine</u>	<u>Description</u>
Field_Get	Returns the details for a field
Field_Set	Sets the details for a field
Field_Save	Saves the field file out to disk
Field_Undo	Undoes changes to field file

## 16.1 Procedure Field\_Get

Returns the details for a field.

<u>Parameters</u>	<u>Description</u>
Fno	Smallint: contains the number of the field
xName	Pointer to ASCIIz string: up to 255 characters long
xStarts	Pointer to ASCIIz string: up to 255 characters long
xEnds	Pointer to ASCIIz string: up to 255 characters long
EmsgBlk	Pointer to Error_Control_Block

If Fno is out of range, Field\_Get returns a null string in xName.

You must supply the buffers for xName, xStarts, and xEnds.

## 16.2 Procedure Field\_Set

Sets the details for a field.

<u>Parameters</u>	<u>Description</u>
Fno	Smallint: contains the number of the field
xName	Pointer to ASCIIz string: up to 255 characters long
xStarts	Pointer to ASCIIz string: up to 255 characters long
xEnds	Pointer to ASCIIz string: up to 255 characters long
EmsgBlk	Pointer to Error_Control_Block

If Fno is zero, field gets added. If xName is NULL, field gets deleted. If Fno is zero AND xName is null, the field gets added and then deleted. This is NOT recommended.

You must supply the buffers for xName, xStarts, and xEnds.

## 16.3 Procedure Field\_Save

Saves the field file out to disk.

<u>Parameters</u>	<u>Description</u>
EmsgBlk	Pointer to Error_Control_Block

## 16.4 Procedure Field\_Undo

Undoes changes to field file.

<u>Parameters</u>	<u>Description</u>
EmsgBlk	Pointer to Error_Control_Block



---

# 17. WYSIWYG DOCUMENTS

ISYS provides an option which allows WYSIWYG browsing of word processor documents. This chapter describes the API that supports that feature.

Code examples are given in either pseudo-code or Visual Basic terms. It is assumed users of other languages will be able to translate to equivalent concepts.

## 17.1 Overview

The standard method for browsing ISYS documents involves the application programmer taking complete responsibility for the user interface. The ISYS engine notionally formats the document into an array of lines containing elementary formatting codes, and the application makes calls to `Get_Line_With_Hit_Before/After` and `Get_Document_Line`, and renders the lines as it sees fit.

The WYSIWYG interface works in a completely different fashion and, in many ways, is easier for the application programmer. After a call to `Open_Document`, the ISYS engine passes an `hWnd` back to the caller, and this `hWnd` becomes an “active player” on the desktop, repainting, scrolling and rendering itself automatically.

Note that whereas ISYS can provide **extremely** high performance browsing in non-WYSIWYG documents, WYSIWYG documents are of necessity slower to browse. In addition to the overhead of rendering images and the like, the WYSIWYG viewers must read the document in logical sequence in order to facilitate the complete formatting context. The non-WYSIWYG browsers, by comparison, can randomly navigate a document of any type with minimal I/O. The WYSIWYG browsers need to load the entire document (up to the visible region) but this is not a prime performance concern unless extremely large documents are involved.

## 17.2 Licensing

Note that the WYSIWYG browsers may be subject to additional licensing considerations depending on the nature of your application, and involve the shipping of additional DLLs and other modules. Contact ISYS Search Software for full licensing details.

The WYSIWYG viewers are licensed from KeyWord Office Technologies, and it is assumed you also have access to the KeyView API documentation.

## 17.3 Creating WYSIWYG Documents

WYSIWYG documents are an “indexing time” option, and full details of how to employ the WYSIWYG option are described in the ISYS Utilities online help. For developers who prefer to directly manipulate the ISYS.CFG file, WYSIWYG indexing rules are the same as normal indexing rules, except they are followed by the keyword “WYSIWYG”.

ISYS only supports WYSIWYG viewing for the following formats: Microsoft Word, WordPerfect, Microsoft RTF, Lotus Ami Pro and Word Pro, HTML, Excel and ASCII.

It is strongly recommended you initially create your WYSIWYG documents using the standard retail version of ISYS:desktop, and verify that you are able to browse them correctly using ISYS:desktop. Indexing WYSIWYG documents through your own application is no different to indexing non-WYSIWYG documents.

## 17.4 Opening a WYSIWYG Document

WYSIWYG documents are opened using the two open document primitives described earlier, `Open_Document` or `Open_Unfound_Document`.

Simply call either entry point in the usual way. However, the `Num_Lines` parameter which usually returns the number of notional “lines” in the document for subsequent access via `Get_Document_Line`, will instead return a negative value representing the `hWnd` of the view window. Applications should set a flag to indicate they are dealing with a WYSIWYG document, and make note of this `hWnd` value to subsequently refer to the view window. Note that you will need to negate the `Num_Lines` parameter in order to turn it into a valid `hWnd`.

The `hWnd` returned by `Open_Document` is an `hWnd` in the `KeyView` sense (refer `KeyView` API documentation), and is actually a handle to an invisible window which controls the view. It is **not** a handle to a visible window. To obtain the handle to the visible window, you need to call the `ISYS WYSIWYG_Multiplex` entry point.

`Open_Document` still returns the usual values through its function result — a negative value representing various error conditions, or a positive value which becomes an ISYS handle to the open document, and is used in other ISYS calls.

If `Open_Document` or `Open_Unfound_Document` returns a negative `Num_Lines`, that means you have opened a WYSIWYG document and should place the `KeyView` browse window into your application using the calls documented here. If `Num_Lines` returns positive, you have opened a normal document, and should display its contents via `Get_Document_Line`.

Once the document is open and you have placed the visible browse window on your application, the window takes care of rendering itself, highlighting hits, scrollbars, etc.

The hWnd passed back in the negated Num\_Lines parameter is not the hWnd of the visible viewer window, but the hWnd of an invisible control window. You can obtain the visible hWnd via a WYSIWYG\_Multiplex call.

## 17.5 The WYSIWYG\_Multiplex call

The ISYS Engine and KeyView viewers are tightly coupled to provide the highlighting of hits, hit navigation, and other functionality present in a text retrieval browser. Much of this functionality is encapsulated and represented through the *WYSIWYG\_Multiplex* call, whose territory is the “gray area” between the ISYS indexing and retrieval Engine and the KeyView WYSIWYG viewers. The definition of this routine is:

<u>Parameters</u>	<u>Description</u>
hDoc	Longint: reference returned when the document was opened
Function	Smallint: function to be performed (documented below)
Result	Pointer to a longint that will contain the result of the function
EmsgBlk	Pointer to Error_Control_Block

Note that the hDoc used in WYSIWYG\_Multiplex is the ISYS document “handle” as returned by the function result from Open\_Document, and as widely used elsewhere in the ISYS API.

The function parameter describes what action you wish to be performed on the document. The result parameter both accepts and value and returns a value. The meaning and usage of the function parameter is described in the following sections, in order of usefulness.

## 17.6 Getting the Visible hWnd

The visible hWnd is retrieved via WYSIWYG\_Multiplex function number 8. The result parameter returns the hWnd of the window that will display the document on the screen.

After you have acquired the handle to the visible window, you will need to set ownership of the window and place the window on the proper background.

You accomplish this by two Windows API calls, the SetParent and the MoveWindow functions. The SetParent call places ownership of the visible window to another window. The second parameter to the SetParent call must be the handle of the window that will be the “parent” of the visible window, typically you own applications’ browser container window.

In doing this, the visible window takes on the properties of the parent window for windows focus and messaging attributes. Once the parent window has been established, the visible window needs to be sized to the area on the parent window that you want it placed with the MoveWindow call. For all messages that the parent window gets on resizing or movement, the KeyView visible window will need a MoveWindow call to properly display the WYSIWYG viewer.

Information on the SetParent and MoveWindow calls is available in your Windows Programmers Reference manual.

#### Code Example

```
hDoc = Open_Document (1, false, num_lines, dtype, pfn, ECB)

if (hDoc >= 0) and (num_lines < 0) then
    ' It is a WYSIWYG document, and it opened ok. Set a
    ' flag so we know elsewhere not to render via
    ' get_document_line, because once we've done this,
    ' rendering will happen for us automatically.

    is_WYSIWYG = true

    ' The handle to the invisible controlling hWnd
    ' was passed back as a negative num_lines

    WYSIWYG_Controlling_hWnd = -(num_lines)

    ' Now get the visible hWnd as well by using Multiplex
    ' function number 8.

    WYSIWYG_Multiplex hDoc, 8, WYSIWYG_Visible_hWnd , ECB

    ' Place the window on our own container window.
    ' Assume our window is called MyForm and we can
    ' Access its hWnd as below

    rc= SetParent (WYSIWYG_Visible_hWnd, MyForm.hWnd)

    ' Make the viewer the appropriate size

    MoveWindow WYSIWYG_Visible_hWnd, 0, 0, width, height
```

In the above pseudo-code fragment, the document is opened in the usual way. If the document opens correctly and is a WYSIWYG document, a flag is set, and note is made of the KeyView controlling (non-visible) hWnd. The multiplex call is then made to determine the visible hWnd.

The SetParent call causes the KeyView viewer window to be placed on your application Window. Most likely, the hWnd you would pass would be the hWnd of your own Form (in Visual Basic terms) or parent Window.

The MoveWindow call positions the view window at the top left, and resizes it to fully occupy the parent window. You would alter this line as appropriate for your parent forms toolbars, etc. In your windows' "resize" event handler, you would also need to perform the same MoveWindow call.

If you are not familiar with the SetParent or MoveWindow Windows API calls, you should seek assistance with the Microsoft Windows API.

These few lines of code immediately give you a full-function WYSIWYG browser. You automatically get all rendering, horizontal and vertical scrolling, repainting, hit highlighting, and so on. In many ways, this is simpler than the standard non-WYSIWYG method where you must provide your own scrollbars and text rendering.

To close a document, first call DestroyWindow passing in the hWnd of the visible Window, then make the ISYS Close\_Document call.

## 17.7 Handling Messages

To make sure the communications between ISYS and the KeyView viewer are working properly, you should echo any messages your parent window receives in the WM\_USER range to the WYSIWYG\_Windowproc function.

The method you use to handle specific Windows messages depends on your programming language, and for Visual Basic, may involve the use of a specialized component such as a message blaster or the CSFORM control. For other languages, this is just a matter of putting the appropriate lines in your Windows message handler.

Whenever you receive a WM\_USER range message, you should pass that message onto ISYS via the WYSIWYG\_Windowproc call:

Function WYSIWYG\_Windowproc:

<u>Parameters</u>	<u>Description</u>
hWnd	Handle to the WYSIWYG_Controlling_hWnd retrieved from the Open_Document or Open_UnFound_Document call.
Msg	Smallint: message you received to pass onto ISYS
wParam	Long: parameter to pass onto ISYS
lParam	Long: parameter to pass onto ISYS
Returns	Long

code example:

```
Function MessageHandler(msg, wParam, lParam)

    if msg > 1024 then goto ISYSmsg

    .

    .

    .

ISYSmsg:

    return_val = WYSIWYG_Windowproc
                (WYSIWYG_Controlling_hWnd, msg, wParam, lParam)

    .

    .
```

The parameters msg, wParam and lParam are all the same parameters you receive in your Windows message handling routine – you are just passing them onto the invisible routine.

## 17.8 Displaying the First Hit

To advance the document to the first hit, call WYSIWYG\_Multiplex with a function code of 7. The third parameter has no significance in this call. If the first hit does not occur on the first visible screen, this call may take a few seconds to execute since the KeyView viewer needs to read ahead into the viewing buffer. If the hit occurs on the first visible screen when the visible window is opened, the screen is not advanced to the next hit.

Assuming the document was opened, the visible hWnd obtained, and the window correctly placed using SetParent and MoveWindow, as above, then the view may be advanced to the first hit as shown below:

**Code Example:**

```
` Document is opened and parent is set

` Now advance to first hit. Temp is a dummy variable

ISYS_Multiplex hDoc, 7, temp, ECB

` First hit is now visible
```

## 17.9 Determining Navigability

The standard KeyView browse window provides standard scrollbars for sequential navigation. However, in the context of an ISYS document, there are other non-sequential aspects to navigation, for example, advancing to the next hit or annotation.

Multiplex call number 5 returns a set of four flags which indicate whether various objects exist before or after the visible portion of the view. In the retail version of ISYS, this call is used to enable and disable the “next hit” and “next annotation” buttons.

When called with a function value of 5, the Result parameter is a bit field interpreted as:

xxx1 - hit follows the currently visible portion

xx1x - hit precedes the currently visible portion

x1xx - annotation follows the currently visible portion

1xxx - annotation precedes the currently visible portion

### Code Example:

```
Dim Result&

` See how our navigation buttons should be set

ISYS_Multiplex hDoc, 5, Result, ECB

Next_Hit.enabled = (result and 1) <> 0

Prev_Hit.enabled = (result and 2) <> 0

Next_Anno.enabled = (result and 3) <> 0

Prev_Anno.enabled = (result and 4) <> 0
```

## 17.10 Navigating

Navigating from hit to hit, or annotation to annotation is achieved simply by making a variety of multiplex calls. These are

function = 1, jumps to next hit following the currently visible portion

function = 2, jumps to hit preceding the currently visible portion

function = 3, jumps to next annotation following the currently visible portion

function = 4, jumps to annotation preceding the currently visible portion

In all cases, the value passed and returned by the Result parameter is ignored. Note the calls which advance to the next hit or annotation may take some time to complete. The KeyView viewers read ahead while the user is inactive, but if it has not yet read ahead sequentially to the destination of the jump, the call may take a second or two to complete.

### Code Example:

```
Dim Dummy&

` Jump to next hit not currently visible

ISYS_Multiplex hDoc, 1, Dummy, ECB
```

Note that you can also make use of the standard KeyView navigation APIs, such as VAPIM\_GOTO\_OFFSET, as described in the KeyView API manual.

## 17.11 Printing

To print the entire document, call WYSIWYG\_Multiplex with a function code of 12. The third parameter has no significance. If the whole document has not been read in yet by the KeyView viewer, there may be some delay in printing until KeyView finishes reading the document into its buffer.

### code example:

```
` if the user clicked on the print button or menu command

WYSIWYG_Multiplex hDoc, 12, temp, ECB
```



## 17.12 Determining how much of the document has been read

To find out how many characters have been read into the KeyView buffer, call WYSIWYG\_Multiplex with a function code of 9. The return variable will tell you how many characters the KeyView viewer has read into its buffer at the time of the function call.

### code example:

```
.
.
.
` determine if we can place an annotation

WYSIWYG_Multiplex hDoc, 9, chars_so_far, ECB

if (annotation.location <= chars_so_far)

    display the annotation on the visible WYSIWYG window

end if

.
.
```

## 17.13 Determining the visible location

To find the first visible offset in the KeyView viewer, call WYSIWYG\_Multiplex with a function code of 13. This is used primary for implementing a “go back” button so you know what offset you were at, and hence what offset to return to. This function should be used after the visible window has received and processed a paint message.

### code example:

```
.
.
.

Dim offset as Long

WYSIWYG_Multiplex hDoc, 13, offset, ECB

.
```

## 17.14 Determining the Page Number

To find out what page an offset value may be on, or what page is currently in view, call WYSIWYG\_Multiplex with a function code of 6.

If the third parameter is -1, this call returns the page number of the top of the current visible window, otherwise it returns the page number for the offset specified in the third parameter. This function should be used after the visible window has received and processed a paint message.

### code example:

```
.  
.   
.   
  
WYSIWYG_Multiplex hDoc, 6, offset, ECB  
  
.   
. 
```

## 17.15 Opening in non-WYSIWYG mode

Sometimes it is desirable to open a WYSIWYG document in non-WYSIWYG mode. For example, you may want to access a few lines from the document invisibly, without reference to a view window.

To open a WYSIWYG document in Get\_Document\_Line mode, call WYSIWYG\_Multiplex with a function code of 10. The first and third parameters are unused.

This function only applies to the next Open\_Document call. Any subsequent calls to Open\_Document will, without using this function again, open the document in WYSIWYG mode. Using this function, you will be able to use the following ISYS API calls on the WYSIWYG document: Get\_Document\_Line, Get\_Line\_With\_Hit\_After, Get\_Line\_With\_Hit\_Before, and Get\_Document\_Line\_Page\_No.

### code example:

```
WYSIWYG_Multiplex 0, 10, temp, ECB
```

## 17.16 WYSIWYG to non-WYSIWYG Function Relationship Chart

	<b>WYSIWYG</b>	<b>non-WYSIWYG</b>
<b>opening a document</b>	Open_Document	Open_Document
<b>viewing a document</b>	WYSIWYG_Multiplex function parameter = 8	up to the programmer
<b>going to the first hit</b>	WYSIWYG_Multiplex function parameter = 7	Get_Line_With_Hit_After LineNo parameter = 0
<b>going to the next hit</b>	WYSIWYG_Multiplex function parameter = 1	Get_Line_With_Hit_After LineNo parameter = current line number
<b>going to the previous hit</b>	WYSIWYG_Multiplex function parameter = 2	Get_Line_With_Hit_Before LineNo parameter = current line number
<b>print entire document</b>	WYSIWYG_Multiplex function parameter = 12	up to the programmer

## 17.17 Other KeyView Features

To implement other abilities of the KeyView API, as documented in the KeyView header and Help files, you will need to use the Window's API call, `SendMessage`.

**code example:**

```
.  
  
    return_val = SendMessage (WYSIWYG_Controlling_hWnd, 1024 +  
707, 0, 0)  
  
.
```

In the above example, the message sent, documented in the KeyView API header files, copies the selected text in the WYSIWYG viewer to the Window's clipboard. The first parameter is the invisible window as returned from the open document calls.

Other functionality provided by addressing the KeyView API directly includes:

- Detection of current position
- Navigation to an arbitrary point
- Determining whether any text is selected
- Copying text to the clipboard
- Changing the mousepointer
- Mouse-click sensing
- Annotation placement
- Annotation sensing
- Search dialogs

Refer to the KeyView API documentation for full details on how to make use of these features. Note that KeyView uses a `SendMessage` oriented API, and the `hWnd` you should send the messages to is the **non-visible controlling hWnd, not the visible hWnd**.

## 17.18 Distributing WYSIWYG Applications

WYSIWYG Applications just require the inclusion of additional DLLs. ISYS:desktop will have installed these DLLs into the VIEWER32 directory beneath the ISYS executable directory. You may place these DLLs correspondingly and the ISYS Engine will find them.

Note, however, that WYSIWYG redistribution and functionality may be subject to additional licensing.



---

# 18. INTELLIGENT AGENT

The ISYS Intelligent Agent API provides access to the underlying Agent machinery. The purpose of this API is to be able to provide users, on a per-user basis, notifications and views of material which is new. That is, material which has entered the ISYS arena and which that particular user has not seen before. Obviously this necessitates storage of what has been seen and what has not.

The application developer may choose to deliver these results through whatever means is deemed appropriate, including push or pull, or any combination of the two. Note that Agent processing may either take place in your thick client application, or centrally. For examples of the Agent API in use, look at the ISYS:desktop and ISYS:web Intelligent Agent features.

In a single-user scenario, ISYS automatically stores Agent management data in a directory tree beneath your Application Data or Windows directory. In a multi-user situation, you need to manage your own multiple storage trees. You may also choose to do this in the case of a single user.

## 18.1 Procedure IA\_Load

Initializes the Intelligent Agents by loading the list of Agents from the default directory location. By default, this will be a directory tree beneath the Application Data or Windows directory, but you may have changed this location using `ISYS_Multi_Set_Agent_Root_Directory`. Where you are running multiple Agents for multiple users, use `ISYS_Multi_Set_Agent_Root_Directory` to set the appropriate root directory for the current user before calling `IA_Load`.

`IA_Load` does not cause individual Agents to perform any processing, nor access their current results. It just loads the directory of Agents, and must be called before any further Agent calls can be made.

If you already have a set of Agents loaded, and change the Agent root directory and call `IA_Load` again, the old list is discarded and the new list is loaded.

<u>Parameters</u>	<u>Description</u>
<code>EmsgBlk</code>	Pointer to <code>Error_Control_Block</code>

## 18.2 Procedure IA\_Save

If you have made changes to the list of Agents, you need to call IA\_Save to write the changes back to disk. You only need to make this call if you have added, deleted or changed an Agent in the list of Agents.

After you have saved your Agent list, the Agent list is still loaded.

The Agent list is saved back according to the default storage scheme, or to the directory root specified in the last ISYS\_Multi\_Set\_Agent\_Root\_Directory call.

<u>Parameters</u>	<u>Description</u>
EmsgBlk	Pointer to Error_Control_Block

## 18.3 Procedure IA\_Close

IA\_Close frees all memory associated with the Agent list currently loaded in memory. It is not necessary to call IA\_Close between IA\_Loads.

<u>Parameters</u>	<u>Description</u>
EmsgBlk	Pointer to Error_Control_Block

## 18.4 Procedure IA\_Count

IA\_Count returns how many Agents are in the currently loaded Agent list.

<u>Parameters</u>	<u>Description</u>
AgentCount	Pointer to a smallint which will be set to the number of Agents in the currently loaded Agent list.
EmsgBlk	Pointer to Error_Control_Block



## 18.5 Procedure IA\_Edit

IA\_Edit lets you add, delete or alter agents in your Agent list.

<u>Parameters</u>	<u>Description</u>
AgentNo	Smallint: Agent number in range 1 to IA_Count.
Action	Char: what action to perform 'G' : gets the details of the chosen Agent 'R' : replaces the details of the chosen Agent 'A' : adds a new Agent (AgentNo ignored) 'D' : deletes the chosen Agent (AgentInfo ignored)
AgentInfo	Pointer to an IA_Record
EmsgBlk	Pointer to Error_Control_Block

IA_Record	
Name	50 character ASCIIz string: descriptive name given to this Agent
Query	200 character ASCIIz string: query command
DBs	200 character ASCIIz string: list of ISYS indexes
Reserved	14 bytes
Filter	Filter_Block: file name filters you wish to apply
LastDone	20 character ASCIIz string: date/time Agent was last executed
Freq	10 character ASCIIz string: how frequently Agent should check '0': Every 15 minutes '1': Every hour '2': Every four hours '3': Every day '4': Every week
Status	200 character ASCIIz string: status message from last execution
NumNew	10 character ASCIIz string: number of <i>new</i> documents found
Qtype	4 character ASCIIz string: how to interpret Query 'C' to interpret Query as a command-based query 'P' to interpret Query as a Plain English Query

## 18.6 Function IA\_Evaluate

This call causes a nominated Agent to check for new information.

If AgentNo is zero, the “most runnable” Agent is checked. That is, the Agent which is most due to be checked. If AgentNo is non-zero, then that particular Agent is checked now, regardless of its specified frequency and when it was last checked.

The function returns non-zero if any new information has been found, otherwise zero.

New information is “queued” for the user to see. Documents are not physically placed in a queue, just references to them. Other entry points exist for maintaining or accessing the queue of unseen results.

To process the results of an evaluation, you need to call IA\_Activate. Note, however, that you do not have to call IA\_Activate immediately. You may choose to call IA\_Activate from a different ISYS instance running in a different process.

Note that Agents do not automatically check for new information asynchronously. You must call IA\_Evaluate periodically to make each Agent perform its processing. You would normally do this from another process separate to your main application process.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
AgentNo	Smallint: Agent number in range 1 to IA_Count.
EmsgBlk	Pointer to Error_Control_Block
<b>Returns</b>	Smallint: non-zero if new information is found

## 18.7 Function IA\_Store\_Size

IA\_Store\_Size returns how many unseen documents are “queued” by a particular agent. Note the documents themselves are not really queued, just references to them as being found by the Agent, but not yet seen by the user.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
AgentNo	Smallint: Agent number in range 1 to IA_Count.
EmsgBlk	Pointer to Error_Control_Block
<b>Returns</b>	Smallint: number of unseen documents

## 18.8 Function IA\_Activate

This entry point is analogous to Perform\_Find, in that it creates a result list based on the previously unseen results found by a given agent. The result list which is loaded is not the complete set of results found by the Agent, but just those which have not previously been “seen” by the user.

The call sets a QRB, similar to Perform\_Find. The call also may cause the currently opened ISYS database to be changed, since the nominated Agent may be supervising a different database to that currently opened.

Once Agent results have been activated, they may be enumerated and processed in the usual way. That is, there is no difference between a result set generated by Agent activation, and one generated by Perform\_Find or Perform\_English\_Find.

Note that the number of documents loaded into the result list and reflected by QRB.Num\_Docs may be less than that indicated by IA\_Store\_Size. This is because possibly a great deal of time may have gone by between the Agent finding the documents, and the result list being activated, and thus the found document may no longer exist in the index. As the result is activated, ISYS checks its index for each document in the list, and removes the document if it is found to no longer exist.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
AgentNo	Smallint: Agent number in range 1 to IA_Count.
HWnd	Handle of a window that will receive any progress messages. This is analogous to the window handle passed in Perform_Find.
QRB	Pointer to a caller-supplied Query_Result_Block which will be filled by the DLL
Database	Pointer to an area of memory which will be filled to the ASCIIz directory name(s) of the ISYS indexes from which the results were activated. After the call, this index will be opened and whatever index was opened previously will be closed.
EmsgBlk	Pointer to Error_Control_Block
<b>Returns</b>	Smallint: non-zero if some of the documents were found to no longer exist

## 18.9 Function IA\_ListForDel

Documents found by an Agent are stored (by reference) in a queue for later activation. The act of activation does not automatically mark a document as “seen”, as the application developer may wish to arrange his or her own paradigm to control removal of documents from the “unseen” list.

Activating a set of Agent results causes the result store to be loaded as a current result set. The result store can be written back to disk using the IA\_Save\_Store\_With\_Dels API, and the IA\_ListForDel controls which documents will be removed from the list during write back.

This gives developers the flexibility to automatically mark as read during activation, unmark, undo, etc.

The Mode parameter controls whether this API adds a document to the list of documents to be removed from the store, removes a document once added, or determines whether a document has been listed for removal from the store.

To mark documents as read, make this call once for each document, passing the document number and a Mode of 1.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
DocNum	Longint: Document number in the range 1 to QRB.Num_Docs
Mode	Smallint: 0: removes the document from the removal list 1: adds the document to the removal list (usual usage) 2: returns whether the document is on the removal list or not.
EmsgBlk	Pointer to Error_Control_Block
<b>Returns</b>	Smallint: when Mode=2, returns zero if the document has not been listed for removal from the unseen document store, or non-zero if it has been listed for removal.

## 18.10 Procedure IA\_Save\_Store\_With\_Dels

Following activation, writes the document store back to disk, removing any document entries which have been listed for deletion with IA\_ListForDel.

Unless you call IA\_Save\_Store\_With\_Dels, your changes queued for action via IA\_ListForDel will not take effect.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
EmsgBlk	Pointer to Error_Control_Block

## 18.11 Procedure IA\_Get\_Next\_Due\_Agent

If you make this call passing an Agent number in the AgentNo parameter, the call returns the date/time that agent is next due to be checked, based on its configured frequency.

If you call with AgentNo set to zero, the call returns with AgentNo set to the number of the most due Agent, and WhenDue set to the date/time that Agent is due to be checked.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
AgentNo	Pointer to a smallint: Agent number in range 1 to IA_Count.
WhenDue	Pointer to a double: date/time the Agent is next due to be checked, expressed as number of days since December 30, 1899, and fractional parts of a day.
EmsgBlk	Pointer to Error_Control_Block



---

# 19. ENTITY RECOGNITION

ISYS version 8 includes the ability to automatically recognize entities occurring in documents – the “who, what and where” involved in a given set of documents that answer a particular query. Entities aid the “discovery” process, sometimes encapsulating the answer the user was really seeking, other times suggesting alternate lines of investigation.

At its simplest, entity recognition entails no more than enabling the option in your ISYS.CFG file before building the index. At a more sophisticated level, you can get involved in dismissing false positives and customizing or replacing the recognition lexicon to reduce false negatives.

Information on customizing the entity recognition process can be found in the ISYS Utilities online help, under Technical Details, ISYS.ELX2 file.

The API provided herein provides facilities for determining what entities have been found in a particular document, or a result set.

Entities are identified uniquely by an Entity ID, which are always four characters in length. Some of the routines provided herein return Entity IDs, and a routine is provided which maps an Entity ID to its details. Entity IDs are unique within an index, but not unique across indexes nor across a chain of indexes.

See also `Get_Document_Line` for information about how recognized entities are presented within the text of the document as you retrieve it at query time.

## 19.1 Entity Names

Entities have the notion of “decorated” or “undecorated” names. For example, “Mr John Smith” is the decorated form of “John Smith”. Similarly, “Worldwide Widgets Inc” is the decorated form of “Worldwide Widgets”.

Sometimes ISYS will distinguish between decorated and undecorated forms, depending on what it sees in the documents. Other times, such distinction makes no sense, for example the concept does not apply to the location “London”.

Wherever ISYS returns an entity name, the name consists of the base name, a character code 01 delimiter, then a character which may either be “>” or “<”, then the decoration if there is one. The character preceding the decoration indicates whether the decoration should go at the front of end of the base entity name.

Examples:

LOS ANGELES

JOHN SMITH #01 < MR

WORLDWIDE WIDGETS #01 > INC

Entities may be recognized in documents in either their decorated or undecorated forms, or both. For example, `Get_Document_Line` may return a line which includes the entity “John Smith”, preceded by the start of entity marker and the unique Entity ID of John Smith. Looking up the details based on the Entity ID will reveal that, although it was written in the document as “John Smith”, the fully decorated form is “Mr John Smith”.

## 19.2 Procedure `Get_Entity_Summary`

This call may be made after a query has been performed, and it returns summarized information about what entities are involved in the result set. If the query spans a chain of indexes, the summary will be normalized across all indexes spanned.

The caller must allocate space for the array of `Entity_Summary_Blocks`, and passes a pointer to that memory in the `ETable` parameter. Before the call, the `Num_Entities` parameter should be set to the maximum number of elements your output array can hold. After the call, it will be set to the number actually used.

Because the result set may span multiple indexes and Entity IDs are only unique within an index, the `EID` field of the `Entity_Summary_Block` is not set.

The time required to generate the summary is proportional to the number of documents surveyed. Usually, the most important entities in a result list will recur multiple times in the “front” portion of the result list. For this reason, you may specify a `MaxDocs` parameter which limits how many documents will be surveyed. A good starting value is 1000. If you want all documents to be surveyed, specify a `MaxDocs` of zero, but be aware this is likely to cause performance problems on very large result sets.

The entity table is populated in order of most frequent entity, so the entities which are the major players in that result list are presented first. If your `ETable` is not large enough to hold all the entities found in `MaxDocs` documents, then the `Num_Entities` most popular will be given.

The surveying of documents takes place according to the current sort order, so it is best to sort your result list prior to calling `Get_Entity_Summary`.

<u>Parameters</u>	<u>Description</u>
<code>MaxDocs</code>	Longint: maximum number of documents to survey
<code>Num_Entities</code>	Pointer to Longint: number of entities
<code>ETable</code>	Pointer to an array of <code>Entity_Summary_Blocks</code>
<code>EmsgBlk</code>	Pointer to <code>Error_Control_Block</code>



**Entity\_Summary\_Block**

EID	4 character Entity ID (fixed length, not zero terminated)
EType	Longint: Entity “type” code (person, organization, etc) ISYS_Entity_Person = 1 ISYS_Entity_Organization = 2 ISYS_Entity_Email_Address = 3 ISYS_Entity_Location = 4 ISYS_Entity_Website = 5
Count	Longint : number encountered in survey of result set
Entity_Name	255 character: ASCIIZ entity name

### 19.3 Procedure Get\_Document\_Entity\_Summary

This call is similar to Get\_Entity\_Summary, but instead of surveying a sample of the result list, produces a comprehensive summary of all the entities recognized within a found document.

The caller must allocate space for the array of Entity\_Summary\_Blocks, and passes a pointer to that memory in the ETable parameter. Before the call, the Num\_Entities parameter should be set to the maximum number of elements your output array can hold. After the call, it will be set to the number actually used.

Entities are placed in the table in order of relative abundance.

<u>Parameters</u>	<u>Description</u>
hDoc	Longint: handle to an open document
Num_Entities	Pointer to Longint: number of entities
ETable	Pointer to an array of Entity_Summary_Blocks
EmsgBlk	Pointer to Error_Control_Block

## 19.4 Procedure Get\_Document\_Entity\_List

Similar to Get\_Document\_Entity\_Summary, but instead of providing a summary of the entities found within a document, provides a detailed list.

Note that the output array has a different structure than for the entity summarization routines.

The caller must allocate space for the array of Entity\_Occurrence\_Blocks, and passes a pointer to that memory in the ETable parameter. Before the call, the Num\_Entities parameter should be set to the maximum number of elements your output array can hold. After the call, it will be set to the number actually used.

Entities are placed in the table in the order in which they occur in the document.

<u>Parameters</u>	<u>Description</u>
hDoc	Longint: handle to an open document
Num_Entities	Pointer to Longint: number of entities
ETable	Pointer to an array of Entity_Occurrence_Blocks
EmsgBlk	Pointer to Error_Control_Block

### Entity\_Occurrence\_Block

EID	4 character Entity ID (fixed length, not zero terminated)
EType	Longint: Entity “type” code (person, organization, etc)
ParaNo	Word : paragraph number of this occurrence
WordNo	Word : word within paragraph number of this occurrence
Entity_Name	255 character: ASCIIZ entity name

## 19.5 Procedure Get\_Document\_Entity\_Record

This call is used to obtain details of an entity, given its Entity ID (for example, as encountered in Get\_Document\_Line).

<b><u>Parameters</u></b>	<b><u>Description</u></b>
hDoc	Longint: handle to an open document
EID	Pointer to 4 character fixed length Entity ID
Ent_Record	Pointer to a single Entity_Summary_Block
EmsgBlk	Pointer to Error_Control_Block

## 19.6 Procedure Get\_Entity\_Type\_Name

Given an entity type code as returned by one of the other entity routines, this call maps the type code into a type name, both plural and singular. For example, entity type 1 might be mapped to 'Person' and 'People'.

The caller must provide pointers which point to a caller-supplied area of memory which will be filled with the descriptive names of the entity type code passed.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
EType	Longint: Entity type code
Singular	Pointer to an area of memory which will be filled to the ASCIIz singular entity type name, for example "Person"
Plural	Pointer to an area of memory which will be filled to the ASCIIz plural entity type name, for example "People"
EmsgBlk	Pointer to Error_Control_Block

## 19.7 Function Get\_Line\_With\_Entity\_After

This call enables navigation from entity to entity within an open document. Pass in the entity ID you wish to find the next occurrence of, and the line number after which it must occur.

<b><u>Parameters</u></b>	<b><u>Description</u></b>
hDoc	Longint: handle to an open document
Entity	Pointer to 4 character fixed length Entity ID
Line_No	Longint: line number after which the desired entity must be found
EmsgBlk	Pointer to Error_Control_Block
<b>Returns</b>	Longint: line number where the entity next occurs, or zero

---

## 20. COM INTERFACE

There are two primary ways of calling the ISYS engine: either a direct call to the DLL as documented in this manual, or calling the engine via COM.

The DLL call interface provides the greatest efficiency and functionality. However, scripting languages such as ASP or Cold Fusion, or tools such as Visual Basic can find it inconvenient to call DLLs, and work much better with COM objects. The ISYS SDK includes a COM interface to commonly-used APIs, which you can use directly from your ASP or Cold Fusion scripts.

Additionally, the COM interface deals with the issue of multi-threading because each instance of the object transparently runs as a separate process, even when created and used by multiple threads in a single process.

Calling the DLL versus instantiating a COM object are two quite different ways of accessing the ISYS engine. You could, if you wanted, mix both methods in a single application. However, the two methods are distinct and there is no interaction between them.

The COM interface is described in the CHM files found alongside this document.



---

## APPENDIX A: DOCUMENT FORMAT CODES

0	Reserved	28	Signature
1	Reserved	29	Mass11
2	ASCII	30	AmiPro
3	WordStar	31	StenoCAT
4	MultiMate	32	Generic
5	MSWord	33	AUTO
6	WordPerfect	34	MSWorks
7	DCA	35	Reserved
8	Source	36	Reserved
9	ProWrite	37	Uniplex
10	WordPerfect42	38	External Access Module DLL
11	XYwrite	39	Transcript
12	WordStar5	40	SGML
13	WordStar2000	41	WYSIWYG
14	Manuscript	42	WinWord 6.0
15	DisplayWrite4	43	Compuserve Mail
16	WangWP	44	ISYS Annotation Files
17	Samna	45	SQL Data Source
18	FW3	46	Windows Write
19	FFT	47	PowerPoint
20	RTF	48	Works 3
21	OK4	49	HTML text only
22	OAI	50	HTML codes revealed
23	dBASEIII	51	PDF (Adobe Acrobat)
24	QA3	52	Excel
25	Word for Windows	53	Email
26	DW5	54	MSG file
27	WangWPplus		

## Appendix C: Document Format Codes

55	AutoCad DWG	71	Generic IFilter
56	Reserved	72	TIFF
57	Lotus WordPro	73	JPEG
58	Subenumerate CHMs	74	ASCII Unicode UTF8
59	Subenumerate ZIPs	75	ASCII Unicode UCS2
60	MP3	76	OpenOffice / StarOffice
61	ACT! Contact Manager	77	XML document file
62	XML record file	78	Freestanding PST
63	Lotus Notes	79	Freestanding Sendmail MBOX
64	EML	80	Microsoft OneNote
65	MHT	81	Microsoft Document Imaging
66	Flash (SWF)	82	Windows Media
67	HTML Metadata only	83	XPS
68	vCard		
69	Reserved		
70	Visio		

Additional codes are defined as the ISYS engine is enhanced to support new file formats, but existing codes never change. In practical terms, the most generally useful format codes are “2” for ASCII, “33” for AUTO, and “38” for the External Access Module DLL.

When ISYS cannot open a particular file (as requested, for example, in function `Auto_Determine_File_Format`), it returns a number not in this list.

Note that codes 58 and 59 are slightly different to the others, in that they do not express a particular file type that ISYS indexes as documents. Rather, these codes act as flags to indicate whether ISYS should *subenumerate* these *container* formats. If these formats are set in your ISYS.CFG, then ISYS will look inside these container formats and index whatever files are found within them, subject to the other formats you have nominated.

In other words, a Word document inside a ZIP file is indexed as document type 25, not document type 59. But unless 59 has been given as an allowable format, the ZIP will not be entered during the document scanning phase.



---

## APPENDIX B: CONSTANTS

These constants make it easier to set parameters to certain values that ISYS wants and makes your code more readable by explaining what they do. Constants are listed with each routine that uses them and are summarized here.

### Open\_Database

These two constants are for procedure **Open\_Database**. They specify the open mode for the database files.

ISYS\_OpenReadOnly = 0

ISYS\_OpenReadWrite = 1

### Perform\_Find and Perform\_SCT\_Find

These constants are for procedure **Perform\_Find** and **Perform\_SCT\_Find**.

ISYS\_Query\_None = 0

ISYS\_Query\_Synonym = 1

ISYS\_Query\_Conflate = 2

ISYS\_Query\_Thesaurus = 8

ISYS\_Query\_Internet\_Syntax = 16

ISYS\_Query\_AND\_With\_Current = 32

ISYS\_SCTFind\_And = 4

ISYS\_SCTFind\_Not = 0

#### Sample code:

```
call Perform_Find "Query", ISYS_Query_None, myhWnd, myQRB,
myMsgBlock

call Perform_SCT_Find MyConceptNum, "Additional Query",
ISYS_Query_Synonym + ISYS_Query_Conflate + ISYS_SCTFind_And,
myhWnd, myQRB, myMsgBlock
```

## Sort\_Documents

These constants are for procedure **Sort\_Documents**.

ISYS\_Sort\_Default = 1

ISYS\_Sort\_NumHits = 2

ISYS\_Sort\_DocSize = 3

ISYS\_Sort\_FilePath = 4

ISYS\_Sort\_FileType = 5

ISYS\_Sort\_FileName = 6

ISYS\_Sort\_DateTime = 7

ISYS\_Sort\_Relevance = 8

ISYS\_Sort\_DocDate = 9

ISYS\_Sort\_Bytes = 10

ISYS\_Sort\_Indexed = 11

ISYS\_Sort\_Title = 12

ISYS\_Sort\_UnPhrasedHits= 13

ISYS\_Sort\_Format = 14

ISYS\_Sort\_Metadata = 15

ISYS\_Sort\_Precedence = 16

ISYS\_Sort\_Cat\_Weighted\_Relv = 17

ISYS\_Sort\_Within\_Index= 32

ISYS\_Sort\_Within\_Category\_Name = 64

ISYS\_Sort\_Within\_Category\_Freq = 128

ISYS\_Sort\_Within\_Category\_Relevance = 192

ISYS\_Sort\_Reverse = -1

## Open\_Document

These constants are the possible return values from **Open\_Document**.

```
ISYS_Open_OK = 0
ISYS_Open_Failed = -1
ISYS_Open_Changed = -2
ISYS_Open_NoRights = -3
ISYS_Open_NoReader = -4
ISYS_Open_Password = -5
ISYS_Open_CacheError = -6
```

## Get\_Document\_Line

These constants are for examining characters returned by **Get\_Document\_Line**.

```
ISYS_Char_SoftSpace = 1
ISYS_Char_Normal = 2
ISYS_Char_Bold = 3
ISYS_Char_Italic = 4
ISYS_Char_Underline = 5
ISYS_Char_Hit = 6
ISYS_Char_Noindexing = 7
ISYS_Char_Paragraph = 16
ISYS_Char_Start_Entity = 17
ISYS_Char_End_Entity = 18
```

## Filename\_Filter

Use these constants in the wAccess field of procedure **Filename\_Filter**.

```
ISYS_Filter_Next = &hF0
ISYS_Filter_This = &hFF
```

Use these constants to enter “don’t care” values for dates, times, and filenames in procedure **Filename\_Filter**.

ISYS\_Filter\_DontCareDate = 0

ISYS\_Filter\_DontCareName = “”

## Set\_Metadata\_Query\_Scope

Use these constants to control which portion of documents are includes in subsequent queries.

ISYS\_MetaScope\_All = 0

ISYS\_MetaScope\_Meta\_Only = 1

ISYS\_MetaScope\_Body\_Only = 2

## Word\_Search

Use these constants to perform a stem search or a sounds-like search with procedure **Word\_Search**.

ISYS\_Wsearch\_Stem = 0

ISYS\_Wsearch\_Sound = 1

## CFG\_Settings and Def\_Settings

Use these constants for procedure **CFG\_Settings** and procedure **Def\_Settings**.

ISYS\_Load = “L”

ISYS\_Save = “S”

## Move\_Rule

Use this constant to delete a rule when calling procedure **Move\_Rule**.

ISYS\_Rule\_Delete = 0

## IDB\_Function

Use these constants when calling **IDB\_Function**. If you are passing several parameters, you must separate them with spaces. For instance,

**Sample code:**

```
call IDB_Function ISYS_IDB_Create + ' ' + ISYS_IDB_Regardless
```

ISYS\_IDB\_Create = “CREATE”

ISYS\_IDB\_Regardless = “REGARDLESS”

ISYS\_IDB\_Update = “UPDATE”

ISYS\_IDB\_Presort = “PRESORT”

ISYS\_IDB\_Rebuild = “REBUILD”

ISYS\_IDB\_Optimize = “OPTIMIZE”

ISYS\_IDB\_Reindex = “REINDEX”

ISYS\_IDB\_Preview = “PREVIEW”

ISYS\_IDB\_Detail = “DETAIL”

ISYS\_IDB\_Statistics = “STATISTICS”

ISYS\_IDB\_Freq = “FREQ”

ISYS\_IDB\_List = “LIST”

ISYS\_IDB\_Alpha = “ALPHA”

ISYS\_IDB\_Reverse = “REVERSE”

ISYS\_IDB\_Common = “COMMON”

ISYS\_IDB\_Memory = “MEMORY”

ISYS\_IDB\_Version = “VERSION”

ISYS\_IDB\_Users = “USERS”

ISYS\_IDB\_Support = "SUPPORT"

ISYS\_IDB\_Check = "CHECK"

ISYS\_IDB\_Dump = "DUMP"  
ISYS\_IDB\_Close = "CLOSE"  
ISYS\_IDB\_Block = "BLOCK"  
ISYS\_IDB\_Error = "ERROR"  
ISYS\_IDB\_Headers = "HEADERS"  
ISYS\_IDB\_Status = "STATUS"  
ISYS\_IDB\_Stats = "STATS"  
ISYS\_IDB\_Formats = "FORMATS"  
ISYS\_IDB\_Options = "OPTIONS"  
ISYS\_IDB\_Tagstats = "TAGSTATS"  
ISYS\_IDB\_Optimize = "OPTIMIZE"  
ISYS\_IDB\_Add = "ADD"  
ISYS\_IDB\_Top = "TOP"  
ISYS\_IDB\_Recategorize = "RECATEGORIZE"  
ISYS\_IDB\_ProcessCachedDeletes = "PROCESSCACHEDDELETES"  
ISYS\_IDB\_RefreshSecurityCache = "REFRESHSECURITY"

## Messages

Use these constants to determine the type of **message** you received back from the DLL.

ISYS\_MSG\_Stream = “M”

ISYS\_MSG\_Titles = “S”

ISYS\_MSG\_Rewrite = “R”

ISYS\_MSG\_Beginscan = “A”

ISYS\_MSG\_Scanprogress = “B”

ISYS\_MSG\_Deindexing = “C”

ISYS\_MSG\_Deleting = “D”

ISYS\_MSG\_Reading = “E”

ISYS\_MSG\_Readprogress1 = “F”

ISYS\_MSG\_Readprogress2 = “H”

ISYS\_MSG\_Updating = “G”

## ISYS\_MSG\_Scanprogress

When the message type was **ISYS\_MSG\_Scanprogress**, compare the integer parameter to these constants to determine the meaning of the longint parameter.

ISYS\_MSG\_Progress\_Dirs = 1

ISYS\_MSG\_Progress\_Files = 2

ISYS\_MSG\_Progress\_Rulefiles = 3

ISYS\_MSG\_Progress\_Index = 4

ISYS\_MSG\_Progress\_Deindex = 5

ISYS\_MSG\_Progress\_Reindex = 6

## Rlist\_Oper

Use these constants when calling **Rlist\_Oper**.

ISYS\_Oper\_And = “AND”

ISYS\_Oper\_Or = “OR”

ISYS\_Oper\_Not = “NOT”

ISYS\_Oper\_Xor = “XOR”

ISYS\_Oper\_Except = “EXCEPT”

ISYS\_Oper\_ButNot = “BUTNOT”

ISYS\_Oper\_Label = “LABEL”

ISYS\_Oper\_In = “IN”

ISYS\_Oper\_ESPIn = “ESPIN”

ISYS\_Oper\_Nearfollow = “..”

ISYS\_Oper\_Farfollow = “...”

## CFG\_Formats

Use these constants when calling **CFG\_Formats**.

ISYS\_CFG\_Write = 1

ISYS\_CFG\_Read = -1

## Fetch\_Rule\_Areas

Use these constants for procedure **Fetch\_Rule\_Areas**. *Multiply* the constant by the rule area you want to use.

ISYS\_Rule\_ThisArea = 0

ISYS\_Rule\_NextArea = -1



## Set\_Concord\_From\_File\_Option

Use these constants to define **Set\_Concord\_From\_File\_Option**.

ISYS\_CCFopt\_Vent = 1

ISYS\_CCFopt\_dBaseID = 2

ISYS\_CCFopt\_dBaseEntire = 4

ISYS\_CCFopt\_ASCII1 = 8

ISYS\_CCFopt\_ASCII2 = 16

ISYS\_CCFopt\_WYSIWYG = 32

ISYS\_CCFopt\_Widelines = 64

ISYS\_CCFopt\_OEM = 128

ISYS\_CCFopt\_HTML = 256

## Get\_Format\_Attribute

Use these constants when calling **Get\_Format\_Attribute**.

ISYS\_Format\_Name = 1

ISYS\_Format\_Code = 2

## Rename\_Indexed\_Document

Use these constants when calling **Rename\_Indexed\_Document**.

ISYS\_Rename\_OK = 0

ISYS\_Rename\_OFN = 1

ISYS\_Rename\_NFN = 2

ISYS\_Rename\_Mode = 3

## ISYS\_Multiplex

Use these constants when calling **ISYS\_Multiplex**.

ISYS\_Multi\_Thresh = 1

ISYS\_Multi\_Ver = 2

ISYS\_Multi\_Line = 3

ISYS\_Multi\_SetMsg = 4

ISYS\_Multi\_TimeStamp = 5

ISYS\_Multi\_DBRoot = 6

ISYS\_Multi\_AuditID = 8

ISYS\_Multi\_Msgbox = 10

ISYS\_Multi\_Conflate = 12

ISYS\_Multi\_PassThruTabs = 14

ISYS\_Multi\_Direct\_IXC = 15

ISYS\_Multi\_Docname\_to\_IXCnumber = 16

ISYS\_Multi\_IXCnumber\_to\_HitNumber = 17

ISYS\_Multi\_IXCnumber\_to\_DocNumber = 18

ISYS\_Multi\_EAM\_Sense\_IXCnumber = 19

ISYS\_Multi\_Get\_Index\_Stats = 20

ISYS\_Multi\_Define\_Syn\_Char = 21

ISYS\_Multi\_DocNumber\_to\_Name = 23

ISYS\_Multi\_Get\_Language\_Code = 24

ISYS\_Multi\_Set\_Indexing\_Date\_Limit = 25

ISYS\_Multi\_Get\_Document\_Access\_Method = 28

ISYS\_Multi\_Set\_Unfound\_Open\_File\_Type = 31

ISYS\_Multi\_Get\_Rlist\_Extent\_For\_Doc = 37

ISYS\_Multi\_Get\_Current\_Instance\_Handle = 40

ISYS\_Multi\_Get\_Current\_Document\_Names = 41

ISYS\_Multi\_Soft\_Fail\_Chain\_Opens = 42

ISYS\_Multi\_Set\_Nonunicode\_Codepage = 44

ISYS\_Multi\_Disable\_Term\_Coloring = 48

ISYS\_Multi\_Line\_Number\_to\_Paragraph\_Number = 49

ISYS\_Multi\_Get\_Paragraph\_Extent = 50

ISYS\_Multi\_Get\_Deferred\_Deletion\_Status = 55

ISYS\_Multi\_Get\_Index\_Status = 56

ISYS\_Multi\_Get\_Term\_Existence\_Map = 61

ISYS\_Multi\_Set\_Fatal\_Error\_Log\_Dir = 62

ISYS\_Multi\_Get\_Index\_Created\_Software\_Ver = 64

ISYS\_Multi\_Get\_Index\_Indexed\_Software\_Ver = 65

ISYS\_Multi\_Disable\_Auto\_Index\_Backups = 66

ISYS\_Multi\_Force\_Auto\_Index\_Backup = 67

ISYS\_Multi\_Get\_Rich\_HTML\_Real\_DocType = 101

ISYS\_Multi\_DocNumber\_to\_associated\_image = 103

ISYS\_Multi\_Add\_Hits\_to\_HTML = 105

ISYS\_Multi\_Set\_OpenUnfound\_Doc\_attributes = 107

ISYS\_Multi\_Set\_Indexing\_Granularity = 110

ISYS\_Multi\_Set\_Instance\_Storage = 112

ISYS\_Multi\_Get\_Instance\_Storage = 113

ISYS\_Multi\_Get\_Document\_Changed\_Status = 115

ISYS\_Multi\_Parse\_Date = 116

ISYS\_Multi\_HTTP\_Get = 118

ISYS\_Multi\_Extract\_Text\_From\_Document = 119

## Entities

Use these constants when dealing with entity types.

ISYS\_Entity\_Person = 1

ISYS\_Entity\_Organization = 2

ISYS\_Entity\_Email\_Address = 3

ISYS\_Entity\_Location = 4

ISYS\_Entity\_Website = 5

---

# APPENDIX C: DATA STRUCTURES

## Action\_Control\_Block

For further information on these fields, please refer to the ISYS Utilities online help, under *Security*, then *Restricting User Access*.

Action Character: Use a constant from the header files:

	<u>Constant</u>	<u>Value</u>	<u>Definition</u>
--	-----------------	--------------	-------------------

	ISYS_Load	"L"	load the structure from SET file
--	-----------	-----	----------------------------------

	ISYS_Save	"S"	save the structure to SET file
--	-----------	-----	--------------------------------

Dir_Constraint	255 character ASCIIz string:	directory constraint
----------------	------------------------------	----------------------

Pr_Line_Limit	Smallint:	printer line limit (0 = no limit)
---------------	-----------	-----------------------------------

DDA_Allowed	Smallint boolean:	DDA enabled
-------------	-------------------	-------------

SynEdit_Allowed	Smallint boolean:	Synonym editing enabled
-----------------	-------------------	-------------------------

LibEdit_Allowed	Smallint boolean:	Saved query library editing enabled
-----------------	-------------------	-------------------------------------

AnnoEdit_Allowed	Smallint boolean:	Annotation editing enabled
------------------	-------------------	----------------------------

Copy_Allowed	Smallint boolean:	File copy enabled
--------------	-------------------	-------------------

Audit_On	Smallint boolean:	Auditing enabled
----------	-------------------	------------------

Audit_UID	255 character ASCIIz string:	user identification string
-----------	------------------------------	----------------------------

Audit_Fname	255 character ASCIIz string:	audit file name
-------------	------------------------------	-----------------

Audit_Errors	Smallint boolean:	audit error messages
--------------	-------------------	----------------------

Audit_Queries	Smallint boolean:	audit queries
---------------	-------------------	---------------

Audit_Results	Smallint boolean:	audit results
---------------	-------------------	---------------

Audit_Browse	Smallint boolean:	audit documents browsed
--------------	-------------------	-------------------------

Audit_Activate	Smallint boolean:	audit documents activated
----------------	-------------------	---------------------------

Audit_Annotate	Smallint boolean:	audit annotation changes
----------------	-------------------	--------------------------

Audit_Starts	Smallint boolean: audit program starts/stops
Audit_Loads	Smallint boolean: audit TSR usage (DOS only)
Audit_Databases	Smallint boolean: audit databases
DirSelect_Allowed	Smallint boolean: allow user to select database
CatEdit_Allowed	Smallint boolean: allow user to edit catalog of databases
Print_Allowed	Smallint boolean: allow user to print selected text
Enforce_Security_Visibility	Smallint boolean: before returning from processing a query, scan the document result list and check that each document will be accessible for the current user. Documents that would not be accessible are transparently removed from the document list before Perform_Find (or any other query method) returns. Applying this setting may have a substantial effect on performance.

## Annotation\_Control\_Block

The format of this control block varies according to the type of annotation. The first nine bytes of the block duplicate the entry in the annotation index and indicate the format of the rest of the block. Because of this structure's complexity, it is defined in syntactically different but structurally equivalent ways for C, Pascal, and Visual Basic.

Row	Longint: line number of the annotation
Col	Longint: column number of the annotation
Typ	Byte: annotation type code (see list in Get_Annotation_Index_Page)
Icon	127-character Pascal string: contains an optional name of an ICO or BMP file. This entry may be null, in which case your program should use a default icon.

The remainder of the structure depends on what it contains.

**Text Annotation:**

Reserved	36 bytes. Call Get_Annotation_Entry_Text
----------	--

**Graphic Annotation:**

FileNameit	255-character Pascal-style string: contains the file name
------------	---

**Hyperactivity:**

Reserved	1 byte contains a type code
Directory	255-character Pascal string: contains DDA script
Command	255-character Pascal string: contains DDA script
Keystrokes	128-character Pascal string: contains DDA script

**Document Link:**

Doclink	255-character Pascal string: contains path and filename of document to open
---------	---

**Query:**

Query	255-character Pascal string: contains ISYS query to execute
-------	---

**Annotation\_Control\_Block (Visual Basic)****Pseudo code:**

```

type Annotation_Control_Block
  Row      As Long
  Col      As Long
  Typ      As String * 1
  icon     As String * 127
  detail   As String * 255
End Type

```

## Annotation\_Control\_Block (Pascal)

### Pseudo code:

```

Type
Annotation_Type = (A_Note, A_Graphic, A_DDA, A_Empty, A_Any);
Annotation_Entry = Record
    row: Longint;
    Col: Longint;
    Typ: Annotation_Type; {byte}
End;

DDA_Page = Record
    Reserved: Byte;
    Dir      : String [255];
    Command  : String [255];
    Keys     : String [128];
End;

Annotation_Detail = record
    icon : String[127];
    Case Integer of
        1: (reserved: string);
        2: (Graph_Name: String[255]);
        3: (DDA: DDA_Page);
        4: (doclink: string [255]);
        5: (query: string[255]);
End;

Annotation_Entry = Record
    idx: Annotation_Entry;
    det: Annotation_Detail;
end;

```



## Annotation\_Control\_Block (C)

### Pseudo code:

```

Annotation_Entry = Record
    row: Longint;
    Col: Longint;
    Typ: Annotation_Type; {byte}
End;

DDA_Page = Record
    Reserved: Byte;
    Dir      : String [255];
    Command  : String [255];
    Keys     : String [128];
End;

Annotation_Detail = record
    icon : String[127];
    Case Integer of
        1: (Note_Size: wordint;
            Num_pages: wordint;
            pages: array [1..16] of integer);
        2: (Graph_Name: String[255]);
        3: (DDA: DDA_Page);
        4: (doclink: string [255]);
        5: (query: string[255]);
    End;
End;

Annotation_Control_Block = Record
    idx: Annotation_Entry;
    det: Annotation_Detail;
end;

```

## Annotation\_Entry

A call to Get\_Annotation\_Index\_Page returns a list of annotations. Each member of the list has this structure:

Row	Longint: line number of the annotation
Col	Longint: column number of the annotation
Typ	Byte: annotation type code

## Category\_Control\_Block

Block that contains the details of a category found by the current query.

Name	ASCIIz: category name (100 bytes)
Num_Docs	Longint: number of documents found in this category
Relevance	Longint: unweighted average relevance of documents found in this category
Weight	Longint: weighing which should be applied to this category

## Concat\_Block

Dir	255 character Pascal string containing the path name of the index.
Num_Docs	Longint: number of documents in the index
Reserved	Longint
Reserved	Longint
Reserved	Longint
Reserved	Longint

## Configuration\_Control\_Block

Action

Character: use a constant from the header files:

Constant	Value	Definition
----------	-------	------------

ISYS_Load	"L"	load the structure from CFG file
-----------	-----	----------------------------------

ISYS_Save	"S"	save the structure to CFG file
-----------	-----	--------------------------------

Name

255-character ASCIIz: index name

Concurrency

Boolean word: indicates index concurrency

Numbers\_Common

Boolean word: indicates that numbers are common

Latency

Smallint: number of minutes to wait after a file has been saved before beginning to index it.

Signif

255-character ASCIIz: significant characters

InSignif

255-character ASCIIz: insignificant characters

AnnoDrive

255-character ASCIIz: AnnotationDrive, blank for default

Title\_Line

Smallint: title line number

Title\_String

100-character ASCIIz: title line contains

Date\_Handling

Boolean word: indicates if intelligent date handling is on

Dot\_Handling

Boolean word: indicates how periods are to be handled

Filename\_Indexing

Boolean word: indicates whether filenames are to be indexed

Annotation\_Indexing

Boolean word: indicates whether annotations are to be indexed

Master\_Doc

255-character ASCIIz: path and file name of master document

Ocr\_Precomp

Boolean word: indicates whether OCR precompensation is on

Number\_Handling

Boolean word: indicates if intelligent number handling is on

Doc\_Profiles

Boolean word: indicates if Word summary information is indexed

LanguageCode

Smallint: the special language interpretation options for this index:

0	None
1	Ignore accents
2	Korean
3	Traditional Chinese
4	Hong Kong Chinese
5	Japanese
6	Arabic
7	Simplified Chinese
8	Cyrillic
9	Greek

10	Turkish
11	Hebrew
12	Vietnamese
13	Baltic
14	Unicode
15	Central European

Indexes which use language code 14 normalize their content into the Unicode character set, and are able to contain multiple disparate languages simultaneously.

Meta_Titles	Boolean word: indicates whether document titles are taken from the document meta data, for formats that explicitly support title fields in their meta data.
Flush_Index_n_Docs	Longint: during an update run, this setting causes the work files to be flushed into the index after a certain number of documents. If this value is set to zero, ISYS will flush as rarely as available resources allow. It is most efficient to flush as rarely as possible, but you may choose to set this to a lower value in some circumstances.
Flush_Index_n_Words	Longint: similar to above, but forces a flush after a certain number of words.
Backup_Generations	SmallInt: determines how many generations of automatic index backup ISYS will generate. A backup generation is taken every time the index is opened with write intentions.
Default_File_Options	SmallInt: controls what indexing options to apply by default to indexing rules, and also apply to non-filesystem-based documents, for example attachments to emails or BLOBs in SQL databases. This parameter is set in the same way as Set_Concord_From_File_Option.
Max_Word_Length	Wordint: controls the maximum significant indexed word length in bytes. Defaults to 20 for ANSI indexes, or 30 for Unicode indexes (since Unicode is represented using UTF8 encoding, words may be longer as measured in bytes). This may be increased to a maximum of 64.
Load_EAM	255-character ASCIIz: name of External Access Module to load, if not ISYSEX32.DLL.
NonUnicode_Codepage	Word: controls what codepage to assume when processing a non-Unicode document format (for example, a plain ASCII file) into a Unicode index.
Deferred_Deleted_Cache_Percent	Boolean word: controls to what extent deleted (or

	changed) documents may have their deindexing deferred until later for efficiency purposes.
Spelling_Tips_Avail	Boolean word: controls whether spelling tip information should be generated during indexing, allowing applications to expose a “Did You Mean?” feature via the Get_Query_Spelling_Suggestion routine.
Cache_Metadata	Boolean word: indicates whether document metadata should be cached into the index so it may subsequently be accessed using Get_Document_Metadata.
Cache_Security	Boolean word: indicates whether document security information should be cached into the index. Where documents exist on a different server to the index, caching security information can great increase performance when Enforce_Security_Visibility is selected.
Index_Type	Byte: used to indicate special purpose indexes
Custom_Props	255-character ASCIIz: special purpose additional information
Entity_Handling	Boolean word: indicates whether automatic entity recognition is enabled.
Cache_Documents	Boolean word: whether document content caching is enabled.

## Chapter\_Entry\_Block

Control block passed to and from Direct\_IXC\_Write and Direct\_IXC\_Read with information about the file.

FilePath	Pascal String of up to 255 characters: contains the path and file name of the file to get or set title information
Title	Pascal String of up to 150 characters: contains the title of the file. If you set the title, you must make sure that the length byte of the Title string is set correctly.
Time_Stamp	Longint: date code for when the file was last modified
FileType	Byte: code for the file type (see section on Low-Level API Calls)
Word_Count	Longint: number of words in the document
Indexed_Words	Longint: number of words from the document that appear in the index file
Option_String	Pascal String [20]: internal
Reserved	15 bytes
Line_Count	Longint (valid for non-WYSIWYG only)
File_Size	Longint
Indexed_Stamp	Longint: date/time code that file was added to index
Text_Date	Longint: first date appearing in body of document
Confirm_Date	Longint: date/time file was last confirmed as being up to date
Non_Unicode_Codepage	Longint: for non-Unicode documents stored in Unicode indexes, this indicates what codepage the document should be read in.
Reserved	8 Bytes
Metadata_Lines	Smallint: number of lines of metadata at the top of the document
ACL_Block	Longint: block number of cached ACL information
Para_Limit_Exceeded	Boolean byte: did document contain more than 65535 paragraphs
Metadata_Stamp	Longint: checksum of injected metadata
Entity_Block	Longint: block number of entities detected

Entity_Count	Longint: number of entities
Para_Count	Word: number of paragraphs
Flags	Byte: associated document existence bitmap
Reserved	24 bytes

## Document\_Control\_Block

The Document Control Block is filled by a call to Get\_Document\_Record.

Document_Num	Longint: index document id. This is passed back for information only. Your application will probably not use this. It represents the unique document number within the index. In the case of a chain of indexes, this is a synthetic number arranged to be temporarily unique throughout the chain.
File_Path	ASCIIz: document name (255 bytes). May be an operating system file name, URL, FTP URL, email message identifier, XML file name and record position, SQL key field, or arbitrary OEM document identifier. The file name is returned in a form suitable for display to the user. Absolute file names are returned in full, while for documents indexed as relative, only the relative portion is returned. If the document name includes any internal information, this is stripped.
Document_Title	ASCIIz: ISYS document title (150 bytes). This defaults to the first text line of the document, but ISYS can be configured to select lines that contain specific strings, or to use the explicit document title described in the document meta data, for formats that support it.
Document_Words	Longint: number of words in document.
Num_Hits	Longint: number of hits in document.
Phrased_Hits	Longint: number of hits, with hits on consecutive words counted as a single hit. This can give a more intuitive hit count where phrases are involved.
Relevance	Longint: an expression of the calculated relevance of this document compared to the other documents in the same result set. Ranges from zero to 100, where 100 is the most relevant. The calculation is based on hit clustering, cluster proximity, document density, word weight and other factors. Relevance is a comparative measure amongst the documents in the same result set.

File_Format	Longint: document format code, as listed in Appendix A.
File_Time_Stamp	Longint: document file date/time stamp or generation counter.
Size_in_Bytes	Longint: document file size in bytes.
When_Indexed	Longint: date/time stamp of when the document was indexed.
When_Confirmed	Longint: date/time the existence of the document was last confirmed (applies to spidered websites only).
Date_in_Document	Longint: the first date appearing in the text of the document.
HTML_File_Format	Longint: in the case of documents which have been dynamically converted to HTML, this shows the true format of the underlying file, as listed in Appendix A.
Chain_Member	Longint: in the case of a chained index, this indicates in which member of the chain the document was found.
SubChain_Document_Num	Longint: in the case of a chained index, this indicates the true (unsynthesised) Document_Num, which is unique within the index, but not necessarily within the chain.
NonUnicode_Codepage	Longint: in the case of non-Unicode documents processed into a Unicode index, this indicates the codepage which was used for this document. The codepage used may have been one explicitly stipulated by the document itself (for example, a CHARSET directive in an HTML page), or one specified to be assumed via the ISYS.CFG, or failing that, the default codepage of the indexing machine
Full_File_Name	ASCIIz: full document name (255 bytes). Fully qualified (even if relative), and including any internal information that would not normally be shown to the user for visual purposes.
Category	ASCIIz: the category to which the document has been assigned.
Metadata_Hits	Longint: the number of hits which occurred within the metadata portion of the document.
Metadata_Lines	Longint: the number of lines of metadata present at the top of the document.
Paragraph_Count	Longint: the number of paragraphs in the document.
Flags	Longint: flags
Entity_Count	Longint: number of entities detected



## EAM\_Context

Info                      Array [1..12] of Byte: internal

## Entity\_Occurrence\_Block

EID                      4 character Entity ID (fixed length, not zero terminated)

EType                   Longint: Entity “type” code (person, organization, etc)

ParaNo                  Word : paragraph number of this occurrence

WordNo                  Word : word within paragraph number of this occurrence

Entity\_Name            255 character: ASCIIZ entity name

## Entity\_Summary\_Block

EID                      4 character Entity ID (fixed length, not zero terminated)

EType                   Longint: Entity “type” code (person, organization, etc)

Count                   Longint : number encountered in survey of result set

Entity\_Name            255 character: ASCIIZ entity name

## Error\_Control\_Block

Control block used to return error messages from ISYS.

Emsg                    ASCIIz string: up to 238 bytes long

MsgID                   Word: error message code

## Filter\_Block

Path_Contains	255-character ASCIIz string: path contains this value
Path_Omits	255-character ASCIIz string: path does not contain this value
Fname_Like	255-character ASCIIz string: file name is like this value
Fname_Unlike	255-character ASCIIz string: file name is not like this value
Category_Like	255-character ASCIIz string: category name is like this value
Category_Unlike	255-character ASCIIz string: category name is not like this
Date_Before_Year	Smallint: four digit year. Document changed time stamp
Date_Before_Month	Smallint: in the range 1–12
Date_Before_Day	Smallint: in the range 1–31
Date_After_Year	Smallint: four digit year
Date_After_Month	Smallint: in the range 1–12
Date_After_Day	Smallint: in the range 1–31
DateInDoc_Before_Year	Smallint: four digit year. First date in the document.
DateInDoc_Before_Month	Smallint: in the range 1–12
DateInDoc_Before_Day	Smallint: in the range 1–31
DateInDoc_After_Year	Smallint: four digit year
DateInDoc_After_Month	Smallint: in the range 1–12
DateInDoc_After_Day	Smallint: in the range 1–31
DateIndexed_Before_Year	Smallint: four digit year. Date document was indexed.
DateIndexed_Before_Month	Smallint: in the range 1–12
DateIndexed_Before_Day	Smallint: in the range 1–31
DateIndexed_After_Year	Smallint: four digit year

DateIndexed_After_Month	Smallint: in the range 1–12
DateIndexed_After_Day	Smallint: in the range 1–31
File Type	Longint: a file format code from Appendix A

## Format\_Rec

Name	40 character ASCIIz: descriptive WP name
Code	20 character ASCIIz: ISYS.CFG keyword
Nominated	Smallint boolean: is format selected or not

## Instance\_Status\_Block

Control block returned by Init\_Instance with information about the installed ISYS Engine:

DLL_Version	16 byte fixed-length string: contains DLL version number
Reserved	Word: Reserved
License_Type	Word: indicates the license type in use
Licensee_ID1	40 byte fixed-length string: the user name you entered when you installed ISYS. Before Init_Instance is called, this field must be loaded with your License Code.
Licensee_ID2	40 byte fixed-length string: the company name you entered when you installed ISYS

## Query\_Result\_Block

Control block returned by Perform\_Find. It contains the results of the search.

## *Appendix D: Constants*

Total_Hits	Longint: number of hits
Diff_Words	Longint: number of different words
Num_Docs	Longint: number of documents
Filtered_Flag	Boolean word: true if result has been filtered
Subquery_Size	Longint: if the query is a sub-query, this value contains the size of the query-set that this query was performed within
Phrased_Hits	Longint: number of hits, with hits on consecutive words counted as a single hit. This can give a more intuitive hit count where phrases are involved.

## Result\_List

A Result\_List (sometimes called rlist\_ptr) type is a pointer to the following structure:

Size	Longint: internal
Words	Word: internal
Suffix	Boolean: internal
Group	Smallint: size of phrase groups
Phwrds	Smallint: number of words in phrase so far
Common	Boolean: internal
Reserved1	Longint: Reserved
Num_Arts	Longint: internal
Reserved2	String of 58 Bytes: Reserved

## SCT\_Block

SCT_Name	ASCIIz string: length 128
Query	ASCIIz string: length 128
ORed	Word boolean: should lower levels be consolidated in queries
Indent	Word: level within the hierarchy (1 to 7)
Info	Array of 4 (array indexes 1..4 for Pascal and Basic and 0..3 for C) strings of ASCIIz string length 70: contains explanatory text
Filter	Filter_Block (see <i>Filename_Filter</i> for format)

## SYN\_Block

Word	ASCIIz string: 64 bytes in length
Next_Syn	Longint: internal
Next_Ring	Longint: internal

## Word\_Pointer\_Block

Block that contains the A-P-W (Article-Paragraph-Word) location of a specific word in a result-list.

Word_Num	Word: contains the word number in the paragraph
Paragraph	Word: contains the paragraph number in the document
Document	Longint: contains the number of the document
Term_Id	Word: term id of this hit. Call Get_Term_ID_Term to map the id into an actual search term from the query

## Word\_Search\_Context

Link	Longint: set to 0 for first call to Word_Search
Filler	76-byte area used by Word_Search

---

# INDEX

## A

Action Control Block .....	78, 225
AnnoIx .....	163, 230
Annotation Control Block .....	164, 226
Annotation Index .....	163, 230
Annotation_Control_Block (C) .....	229
Annotation_Control_Block (Pascal) .....	228
Annotation_Control_Block (Visual Basic) .....	227
Annotations .....	161
ASCIIz .....	27
Attach_Custom_Relevance_Rating .....	146
Auditing .....	134
Auth_Activate .....	124
Auth_Deactivate .....	125
Auth_GetLastError .....	126
Auth_GetSystemName .....	126
Auth_Initialize .....	124
Auth_Openable .....	125
Auth_Uninitialize .....	124
Authorization Codes .....	30
Auto_Determine_File_Format .....	62

## B

Backups .....	76
Bold .....	48, 117

## C

C language .....	27
Cache documents .....	76
Callback .....	110, 144, 145
Categories .....	65
Category Control Block .....	65, 230
CFG_Formats .....	77
CFG_Settings .....	74
Changed Documents .....	142
Character Format Codes .....	117
Character Sets .....	23
Close_Concording_Run .....	94
Close_Database .....	70
Close_Document .....	50
Close_Find .....	69
Close_Instance .....	50
Close_Rules .....	81
COM .....	20
COM Interface .....	18, 144

Combining results .....	152
Common .....	118
Common, IDB_Function .....	88
CommonThreshold .....	133
Complete_DeConcording .....	95
Concord_From_File .....	93
Configuration Control Block .....	74, 231
Conflation .....	135
Constants .....	213
Create, IDB_Function .....	86
Create_Database .....	93
Create_Rule .....	81

## D

Data Structures .....	225
Date Parsing .....	142
Date/Time format .....	101
DeConcord_From_File .....	95
Def_Settings .....	78
Details, document .....	40
Diacritical marks .....	24
Direct_IXC_Read .....	129
Direct_IXC_Write .....	129
Distributing applications .....	19
DLL .....	13, 15, 25, 31, 108, 145, 155, 239
DocBlk .....	41, 235
Document Control Block .....	41, 235
Document Control Record .....	234
Document list .....	38
Document management system .....	110, 152
Document-control system .....	91
DOS .....	54, 93, 109, 114

## E

EmsgBlk .....	29, 237
Entities .....	48, 76
Entity Recognition .....	203
Error Message .....	28
European languages .....	24
ExistOnPath .....	52
Ext_Auto_Format .....	121
Ext_Close_Document .....	119
Ext_Finalize .....	120
Ext_Foreign_File_System .....	115
Ext_Get_File_Size .....	116
Ext_Get_Names .....	121

Ext_Get_Time_Stamp .....	116
Ext_Initialize .....	119
Ext_Open_Document .....	115
Ext_Prepare_For_WEP .....	120
Ext_Read_Character .....	117
Ext_Release_Document .....	114
Ext_Request_Document .....	114
Ext_Scan_First_Directory .....	113
Ext_Scan_Next_Directory .....	113
Ext_Seek .....	119
External access module .....	92

**F**

Fetch_Rule_Areas .....	80
Fetch_Rule_List .....	80
Field_Get .....	179
Field_Save .....	180
Field_Set .....	180
Field_Undo .....	180
Fielded Searching .....	70
File name .....	52
Filename_Filter .....	52
Filter .....	52
Filter Block .....	53, 238
Find_Similar .....	65
Foreign languages .....	151
Foreignlanguages .....	136
Format Codes, Character .....	117
Format Entry .....	77, 239
Freq, IDB_Function .....	88
Functions .....	27

**G**

Get_Annotation_Entry .....	164
Get_Annotation_Entry_Text .....	166
Get_Annotation_Index_Page .....	163
Get_Annotations_Changed_Status .....	167
Get_Category_Record .....	65
Get_Database_Name .....	34
Get_Document_Entity_List .....	206
Get_Document_Entity_Record .....	207
Get_Document_Entity_Summary .....	205
Get_Document_Group_Entry .....	66
Get_Document_Id .....	132
Get_Document_Line .....	47
Get_Document_Line_Page_No .....	60
Get_Document_Metadata .....	67
Get_Document_Record .....	40
Get_Document_Security_Descriptor .....	149
Get_Document_Time_Stamp .....	60
Get_EAM_Context_Info_for_Hit .....	61

Get_Entity_Summary .....	204
Get_Entity_Type_Name .....	207
Get_Format_Attribute .....	59
Get_Hit_Pointer .....	58
Get_Line_With_Entity_After .....	208
Get_Line_With_Hit_After .....	45
Get_Line_With_Hit_Before .....	45
Get_Line_With_Term_After .....	64
Get_Line_With_Term_Before .....	64
Get_Num_Processors .....	149
Get_Query_Spelling_Suggestion .....	67
Get_Term_ID_Term .....	64
Granularity .....	141

**H**

Hard return .....	117
Hit style .....	48
HTML .....	140, 141
HTTP .....	142

**I**

IA_Activate .....	199
IA_Close .....	196, 207
IA_Count .....	196, 208
IA_Edit .....	197
IA_Evaluate .....	198
IA_Get_Next_Due_Agent .....	201
IA_ListForDel .....	200
IA_Load .....	195, 206
IA_Record .....	197
IA_Save .....	196
IA_Save_Store_With_Dels .....	201
IA_Store_Size .....	198
IDB_Function .....	83
IFilter .....	122
Ignoring text .....	118
Images .....	140
Indexing API .....	20
Indexingfilter .....	136
Init_Instance .....	31
Instance .....	144
InstBlk .....	32, 239
Intelligent Agent .....	141, 195
Interrupt_Processing .....	58
ISYS_Multiplex .....	133
ISYSAUTH.DLL .....	20
ISYSLN.DLL .....	19
ISYSPDF3.DLL .....	19
ISYSPDFL.DLL .....	19
ISYSU732.DLL .....	19
ISYSWNET.DLL .....	19



Italic.....	48, 117	Pascal strings.....	27
IXC Reading.....	135	PDF.....	19
<b>J</b>		Perform_English_Find.....	37
Japanese.....	24	Perform_Find.....	35
<b>K</b>		Perform_SCT_Find.....	173
KeyView.....	181	Preview, IDB_Function.....	87
Korean.....	24	Procedures.....	27
<b>L</b>		Progress.....	33
Language.....	75	<b>Q</b>	
Linlength.....	134	Query.....	35
List, IDB_Function.....	88	QueryResultBlock.....	36, 239
LOAD directive.....	109	<b>R</b>	
Loading your EAM.....	108	Real time.....	91
Longint.....	27	Redistribution.....	19
Lotus Notes.....	19	Reindex, IDB_Function.....	87
<b>M</b>		Relevance.....	68
MBCS EAMs.....	118	Remove_Document_Record.....	58
Merge_Index.....	147	Resource limit.....	56
Merging results.....	152	Result Blocks.....	28
Message Handling.....	134	Result List.....	156, 240
Messageboxes.....	134	Retrieval API.....	21
Messages.....	33	Return, hard.....	117
Messages with Visual Basic.....	23	Return, soft.....	117
Metadata.....	67, 76	Rlist.....	156, 240
Move_Rule.....	81	Rlist API Calls.....	155
Multiple Instances.....	143	Rlist_Append_Entry.....	157
Multiplex.....	133	Rlist_Copy_From_To.....	160
Multiprocessing.....	149	Rlist_Create.....	155
MyFilteringCallback.....	146	Rlist_Current_Rlist.....	158
MyIndexingCallback.....	145	Rlist_Drop.....	156
MyRelevanceRanker.....	146	Rlist_Empty.....	158
<b>N</b>		Rlist_Exchange.....	160
Names, database.....	34	Rlist_Get_Entry.....	156
Naming your EAM.....	108	Rlist_Get_QRB.....	157
Normal style.....	48, 117	Rlist_Oper.....	159
<b>O</b>		Rlists Allocation and Deallocation.....	153
Open_Database.....	34	Rlists Data Structure.....	153
Open_Document.....	44	Rlists Manipulations.....	153
Open_Rules.....	79	<b>S</b>	
Open_Unfound_Document.....	63, 141	SCT_Close.....	172
OPTIMIZE, IDB_Function.....	87	SCT_Close_and_Save.....	172
<b>P</b>		SCT_Entry.....	170
Paragraph marker.....	48	SCT_Insert_After.....	171
Parameters.....	27	SCT_Move.....	172
Pascal.....	27	SCT_Open.....	170
		SCT_Replace.....	171
		SCT_Set_File_Name.....	174
		SCTBlock.....	171, 240
		Security.....	20, 76, 123, 149

Set_Annotation_Entry .....	165	Synonyms .....	136
Set_Annotation_Entry_Text .....	166		
Set_API_Charset .....	147	<b>T</b>	
Set_Concord_From_File_Option .....	96, 97	TAB .....	135
Set_Engine_Call_Tracing .....	150	Thesaurus .....	19
Set_Indexing_Callback_Hook .....	144	Threadsafe .....	144
Set_Indexing_Filter_Hook .....	145	Titles .....	75
Set_Maximum_Found_Documents .....	56	Translate_Hyperstring .....	62
Set_Metadata_Query_Scope .....	68		
Set_Metadata_Relevance_Boost .....	68	<b>U</b>	
Set_Metadata_Sort_Key .....	69	Underline .....	48, 117
Set_Num_Indexing_Processes .....	150	Unicode .....	22, 27, 42, 106, 118, 147
Set_Rlist_Curfew .....	56	Update, IDB_Function .....	86
Set_Sub_Find .....	55	Users, IDB_Function .....	89
Smallint .....	26	UTF-8 .....	22, 27, 147
Soft Return .....	117	UTF8_to_Widechar .....	148
Soft space .....	117		
Sort .....	38	<b>V</b>	
Sort_Documents .....	38	Verb conflation .....	36
Spelling tips .....	67, 76	Version, Database .....	133
SQL .....	19	Version, IDB_Function .....	89
Start_Concording_Run .....	93	Visual Basic .....	27
Statistics .....	136		
Statistics, IDB_Function .....	87	<b>W</b>	
Strings .....	27	WEP .....	119, 120
Support, IDB_Function .....	89, 90	Widechar_to_UTF8 .....	148
Switch_Instance .....	143	WIDELINES .....	47
SYN_Add_Word_To_Ring .....	177	Windows .....	13, 110, 120
SYN_Delete_Entry .....	177	Word length .....	76
SYN_Get_Entry .....	176	Word_Pointer_Type .....	157, 241
SYN_Get_Head .....	176	Word_Search .....	57
SYN_Replace_Word .....	176	Word_Search_Context .....	57, 241
SYN_Save .....	177	Write_Back_Annotations .....	167
SYN_Type .....	175, 241	WYSIWYG .....	21, 181
SYN_Undo .....	177	WYSIWYG_Multiplex .....	181
Synonym expansion .....	36		