Adabas Text Retrieval

Reference Manual for UNIX and Windows

SOFTWARE AG®

Manual Order Number: TRS232-030UNW

This document applies to Adabas Text Retrieval version 2.3.2 for UNIX and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Readers' comments are welcomed. Comments may be addressed to the Documentation Department at the address on the back cover or to the following e-mail address:

Documentation@softwareag.com

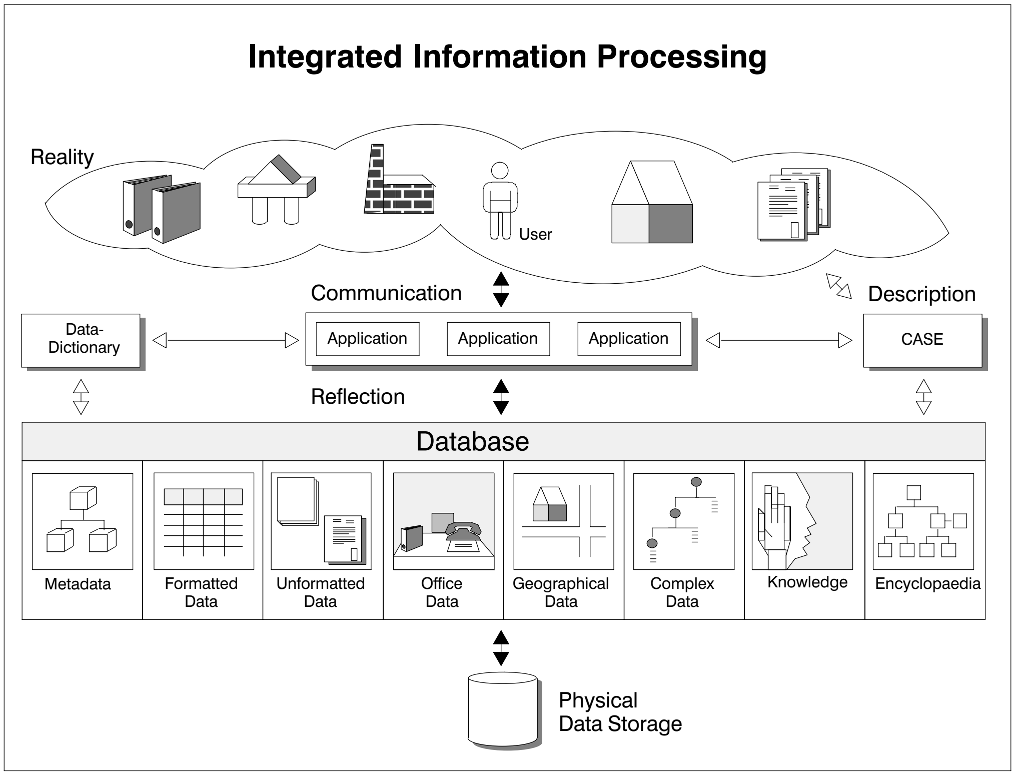
TABLE OF CONTENTS

INTRODUCTION

In industrialized societies, where the quantity of available information is increasing exponentially, a major priority has become effective information management and distribution. Database technology provides perhaps the only means of managing information of such vast proportions.

Traditional information processing has been performed almost exclusively on *formatted data* — data having a specified type and length. Advances in software and hardware technology, however, have made possible the storage and retrieval of textual information as well. The ability to process unformatted data makes it possible to extract needed information quickly from a large text data bases.

The demand for efficient text retrieval is large and growing rapidly in almost all industries. For example, textual information is found in great quantities in such fields as publishing, library archiving, law and technical documentation. All of the necessary data retrieval and data management services needed to create a comprehensive and truly integrated information processing environment are provided by Adabas Text Retrieval together with other Software AG products.


Integrated Information Processing
Reality
User
Communication
Description
Data-Dictionary
Application
Application
Application
CASE
Reflection
Database
Metadata
Formatted Data
Unformatted Data
Office Data
Geographical Data
Complex Data
Knowledge
Encyclopaedia
Physical Data Storage

Adabas Text Retrieval Overview

Adabas Text Retrieval is the heart of Software AG's text retrieval architecture. It offers the full range of functionality expected of powerful information retrieval systems. Applications which access both formatted and unformatted data simultaneously can be developed using Adabas Text Retrieval. Other Software AG products which apply this architecture include:

- Natural Document Management — a complete document management system;
- Con-nect Document Retrieval — a optional extension to the functionality of Software AG's office information system Con-nect.

Since Adabas Text Retrieval is an extension of Software AG's database management system Adabas, it inherits such advantages as high-performance data compression, on to restart, automatic recovery and 24-hour operation.

Adabas Text Retrieval manages the index information and not the content of the data. This means that document contents can be stored at any location (Adabas, sequential files, CD-ROM, PC, etc.).

Adabas Text Retrieval can be used via its call interface from inside Natural or any third generation language such as COBOL or PL/1.

Adabas Text Retrieval Functionality

Text can be designated either as formatted or unformatted depending on your requirements. Unformatted text is referred to in the remainder of this manual as free-text chapters.

Free-text chapters are subject to a process called inversion which creates the information necessary to retrieve a text based on content. Any of three different *inversion* methods can be used:

- Full-text inversion;
- Thesaurus-controlled inversion;
- Inversion using stopword lists.

Numerous functions and operators are available for flexible retrieval:

- Word searches
- Word truncation:
  - Right truncation;
  - Left truncation;
  - Left and right truncation:
  - Middle truncation.
- Phonetic searches;
- Synonym searches;
- Integration of thesaurus relations:
  - Broader terms;
  - Narrower terms;
  - Synonyms.
- Relational operators;
- Boolean operators;
- Structure-independent search (any combination of free-text chapters and formatted fields);
- References to previous queries (refinement);
- Sorting in ascending and descending order;
- Highlighting of found items.

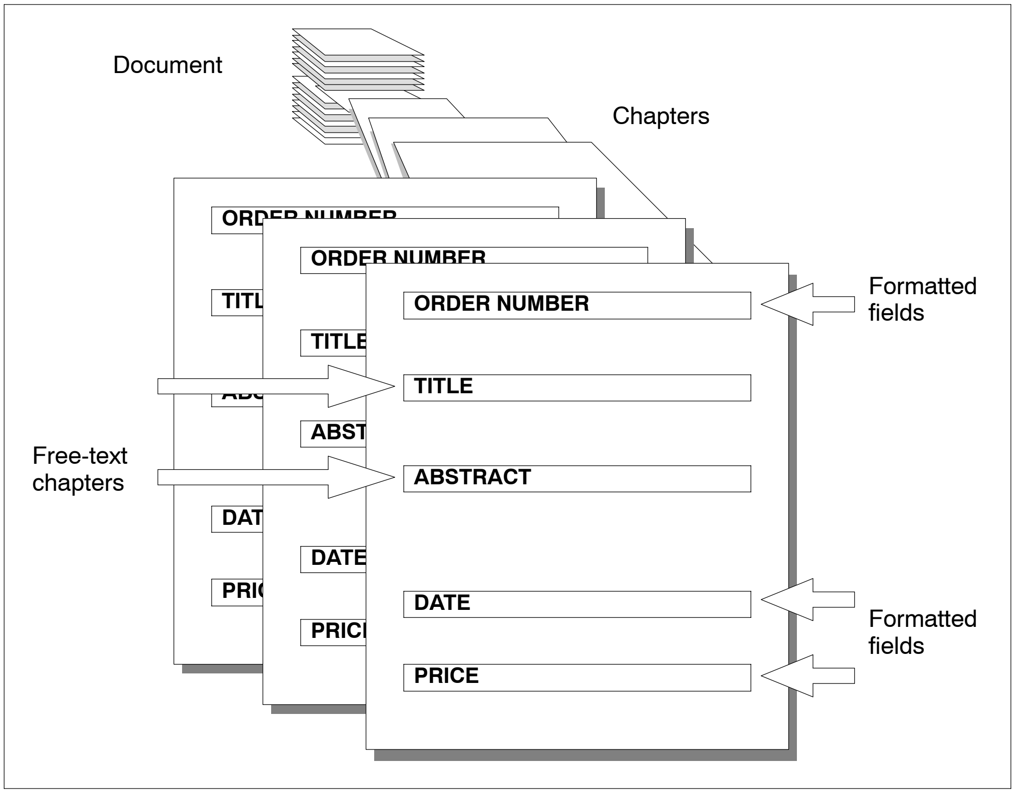
Adabas Text Retrieval Terminology

Defined below are the most important and frequently used terms in Adabas Text Retrieval.

Document

Documents consists of chapters (sometimes referred to as categories) which are equivalent to fields in the relational database model. Chapters can either be designated as free-text chapters, which are managed by Adabas Text Retrieval, or as formatted fields in accordance with the database system.

Free-text chapters can be separated into paragraphs and sentences. This allows you to issue queries which search individual sentences and paragraphs.



Inversion

Inversion is the process which creates the necessary document index entries for the contents of free-text chapters in the document. Adabas Text Retrieval supports three inversion methods:

- Full-text inversion;
- Inversion using a controlled thesaurus;
- Inversion by ignoring words in the stopword list.

You can choose one of the inversion methods for each free-text chapter.

ADABAS TEXT RETRIEVAL CALLS

Adabas Text Retrieval provides calls which perform the following functions:

- Set up sessions;
- Invert free-text chapters;
- Retrieve text and information;
- Browse through ISN sets;
- Highlight search terms;
- Invoke a tokenization process.

This chapter explains what each call is designed to carry out and the parameters of each call. The names of dynamic parameters appear in uppercase capital letters. All other parameters appear in italics.

Alphabetical Listing

The following table lists all available Adabas Text Retrieval calls in alphabetical order:

Call	Description	Page
ADD	Creates document index entries	10
BC	Starts a session	14
CL	Closes a session	16
DDS	Deletes document index entries	17
DSL	Defines search labels	19
DYP	Changes dynamic parameters	21
EISE	Ends browsing through an ISN set	23
EISG	Browses through an ISN set	25
EISS	Starts browsing through an ISN set	27
HIGH	Highlights a document	29
PHON	Translates a token to a phonetic value	32
QR	Executes a query	34
RQR	Releases a query	37
RULE	Defines inversion rules	38
SCA	Scans a free-text line and returns a token	77
SCTC	Defines acharacter table	62
SCTS	Defines classes of character	61
SCTT	Defines a translation table	64
SCTW	Defines reserved words	67
SCTX	Defines the tokenization logic	71

Topical Listing

The following table provides a cross reference of Adabas Text Retrieval calls according to function:

Topic	**Calls**	**Page**
Setting up Sessions	BC	14
	CL	16
	DYP	21
Inverting Documents	RULE	38
	ADD	10
	DDS	17
	PHON	32
Retrieving Text and Information	DSL	19
	QR	34
	RQR	37
Browsing through ISN Sets	EISE	23
	EISG	25
	EISS	27
Highlighting	HIGH	29
Invoking a tokenization process	SCA	77
	SCTC	62
	SCTS	61
	SCTT	64
	SCTW	67
	SCTX	71

ADD

Description

The ADD call creates the document index entries for the contents of a free-text chapter within a document. This process is called document inversion.

Before the ADD call can be executed, the free-text chapter to which the entered text belongs must be established.

The free-text chapter is established by a BC or DYP call which provides the name of the Adabas hyperdescriptor associated with the free-text chapter in question, as the value of the TEXT parameter.

The ADD call stores entries in the document index; it does not store the document text.

Example

See page 153.

Call Format

```
CALL  'TRS'  'ADD'  parameters
```

Required Parameters	Format	Length	In/Output
Return Code	binary	4 bytes	output
Document-ID	alphanumeric	variable	input
Source-Text Length	binary	4 bytes	input
Source Text	alphanumeric	variable	input
Document ISN	binary	4 bytes	output
End-of-Text Indicator	alphanumeric	6 bytes	input

Return Code

The return code is the message delivered at the end of processing which indicates whether an error has occurred. A zero code indicates the normal end of processing. Other codes are explained in Appendix A **Messages and Codes**.

Document-ID

A unique Document-ID must be provided. There are two different ways of providing a Document-ID depending on the setting of the dynamic parameter DOCID of the DYP call:

- If the DOCID parameter contains the name of a formatted field (Adabas descriptor), the *Document-ID* parameter must contain a unique value for the formatted field in question. If a record containing the specified value of *Document-ID* already exists, the Adabas ISN of this record is used in the document index, otherwise an Adabas record containing only the value for the formatted field in question is added to the document file.
- If the DOCID parameter contains the value '##', the *Document-ID* parameter must contain the Adabas ISN of a record on the document file. An Adabas record with the specified ISN must already exist on the document file.

Source-Text Length

The length of the text in bytes, as contained in the parameter *Source Text*.

Source Text

The text to be inverted.

Document ISN

The Document ISN reflects either of the following ISNs:

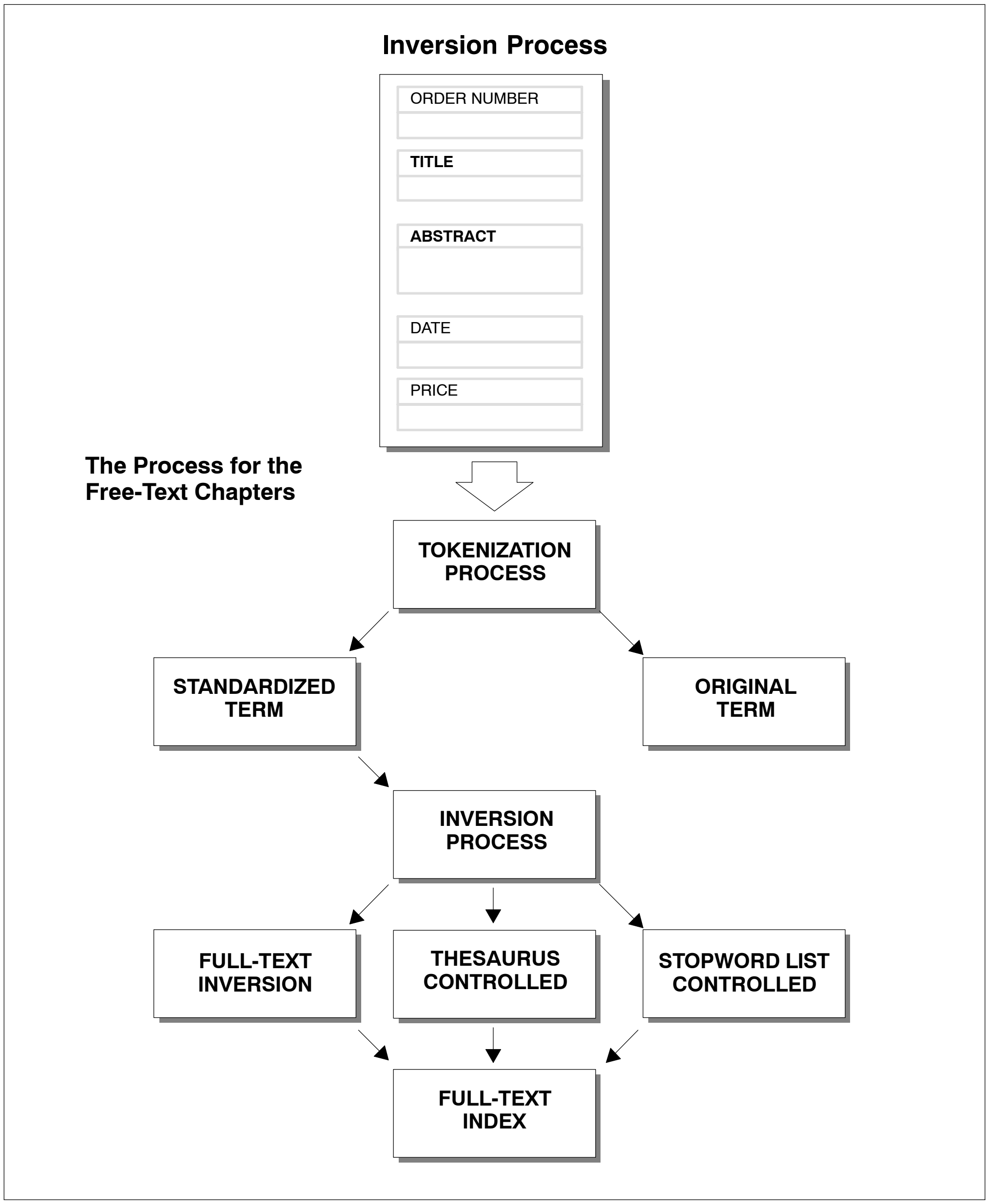
- the ISN entered in the Document-ID parameter;
- the ISN of the record containing the value of the Document-ID parameter.

End-of-Text Indicator

Free-text chapters can be inverted as one text string or divided into several parts. The *End-of-Text Indicator* parameter must contain either of the following values:

LAST	For the last part of a free-text chapter.
NOLAST	In all remaining cases.

The inversion process for the contents of a free-text chapter can generally be executed in one step. However, for the inversion of long texts it may be necessary to execute the inversion in multiple steps, because intermediate Adabas end transactions may be required in order to prevent an Adabas Hold Queue overflow.


Inversion Process
ORDER NUMBER
TITLE
ABSTRACT
DATE
PRICE
The Process for the Free-Text Chapters
TOKENIZATION PROCESS
STANDARDIZED TERM
ORIGINAL TERM
INVERSION PROCESS
FULL-TEXT INVERSION
THESAURUS CONTROLLED
STOPWORD LIST CONTROLLED
FULL-TEXT INDEX

BC

Description

The BC call opens an Adabas Text Retrieval session. This call is mandatory and must be invoked once at the beginning of each session.

Example

See page 142.

Call Format

```
Call   'TRS'   'BC'   parameters
```

Required Parameters	Format	Length	In/Output
Return Code	binary	4 bytes	output
Size of Buffer	binary	4 bytes	in/output
Save Area	alphanumeric	100 bytes	output
Dynamic Parameters	alphanumeric	variable	input

Return Code

The return code is the message delivered at the end of processing which indicates whether an error has occurred. A zero code indicates the normal end of processing. Other codes are explained in Appendix A **Messages and Codes**.

Size of Buffer

The size of the Adabas Text Retrieval buffer.

Minimum Size	32 K
Recommended Size	64 K

Save Area

The name of the storage area used by Adabas Text Retrieval as a temporary save area.

Dynamic Parameters

Any of the following dynamic parameters can be specified in a BC call:

AUTOASP	**DSFNR**	**INDEX**	**SETCHAR**
CONCHAR	**ERRADA**	**MAXDPRO**	**TEXT**
DBTYPE	**ERRPRE**	**MAXVSET**	**TRUNCHAR**
DEFOPER	**HIGHLIGHT**	**MULTICALL**	**VFNR**
DFNR	**HOLDWORD**	**PASSWORD**	**WORDLEN**
DOCID	**INCVOC**	**SEARCHLB**	

For details of the dynamic parameters and their possible values, refer to the section entitled **Dynamic Parameters for the DYP and BC Calls** on page 40.

CL

Description

The CL call closes an Adabas Text Retrieval session and releases all resources.

Example

See page 144.

Call Format

```
CALL 'TRS' 'CL' parameters
```

Required Parameters	Format	Length	In/Output
Return Code	binary	4 bytes	output
Save Area	alphanumeric	100 bytes	output

Return Code

```
Return Code
```

Save Area

This parameter refers to the storage area used by Adabas Text Retrieval for a temporary save area.

DDS

Description

The DDS call deletes the document index entries for a specific free-text chapter within a document.

Important

Before the DDS call can be executed, the free-text chapter to which the entered text belongs must have been established.

The free-text chapter is established by a BC or DYP call which provides the name of the Adabas hyperdescriptor associated with the free-text chapter in question, as the value of the TEXT parameter.

The DDS call deletes entries in the document index; it does not delete the document text.

Example

See page 155.

Call Format

```
CALL  'TRS'  'DDS'  parameters
```

Required Parameters	Format	Length	In/Output
Return Code	binary	4 bytes	output
Document-ID	alphanumeric	variable	input
Delete Option	alphanumeric	3 bytes	input

Return Code

The return code is the message delivered at the end of processing which indicates whether an error has occurred. A zero code indicates the normal end of processing. Other codes are explained in Appendix A **Messages and Codes**.

Document-ID

A unique Document-ID must be provided. There are two different ways of providing a Document-ID depending on the setting of the DOCID parameter of the DYP call:

- If the DOCID parameter contains the name of a formatted field (Adabas descriptor), the *Document-ID* parameter must contain a unique value for the formatted field in question.
- If the DOCID parameter contains the value '##', the *Document-ID* parameter must contain the Adabas ISN of a record on the document file.

Delete Option

There is only one possible delete option:

SUM	Deletes the document index entries for the current free-text chapter.

Note:
If more than one free-text chapter exists for a document whose index entries are to be deleted, the document index entries for each chapter must be deleted separately using repeated pairs of DYP and DDS calls.

DSL

Description

The DSL call defines search labels on the document index. These labels allow direct referencing of both formatted fields and free-text chapters in queries.

For free-text chapters, the same search label can be entered for more than one Adabas hyperdescriptor name, thus making up a global search label which enables the user to address multiple free-text chapters with one search label in a query.

Example

See page 143.

Call Format

```
CALL 'TRS' 'DSL' parameters
```

Required Parameters	Format	Length	In/Output
Return Code	binary	4 bytes	output
Search Labels	alphanumeric	variable	input

Return Code

The return code is the message delivered at the end of processing which indicates whether an error has occurred. A zero code indicates the normal end of processing. Other codes are explained in Appendix A **Messages and Codes**.

Search Labels

A search label can be assigned to an Adabas field. This can either be a formatted field (Adabas descriptor) or a free-text chapter (Adabas hyperdescriptor).

The definition of search labels consists of a character string containing one or more entries separated by commas and ending with a period.

To define search labels, use the following syntax:

'Y1Y1=name1,AA=name2,Y2Y2=name3,AB=name4[,...].'
'Y1Y1=(name1,thesname)[,...].'
'Y1Y1=name1,Y2Y2=name2(,...).'

where *Y1Y1* and *Y2Y2* are text inverted fields and *AA* and *AB* are formatted inverted fields.

name1-4 are logical names for text inverted and formatted inverted fields. IN order to use logical names for retrieval of text inverted and formatted inverted fields in the document structure, search-label names are assigned to these fields.

thesname is a placeholder for a thesaurus linked to a field. Thesaurus names must not exceed 8 bytes.

Example:

'Y1Y1=TITLE, Y2Y2=ABSTRACT, AH=NUM,AI=DATE,Y1Y1=ABSTI, Y2Y2=ABSTI.'

---

where TITLE and ABSTRACT are the *search labels* for text inverted fields.
where NUM and DATE are the *search labels* for formatted inverted fields.
where ABSTI is the *global search label* for text inverted fields.

For information about *search label* and *global search label*, see Chapter **Query Syntax**.

Note:
Formatted inverted fields are specified as two-character bytes. Text inverted fields are specified as four-character bytes.

DYP

Description

The DYP call enables users to define dynamic parameters or redefine any parameters specified in the BC call at the start of a session or in previous DYP calls. For example, the call enables users to handle more than one free-text chapter within a document by changing the TEXT parameter.

Example

See pages 153, 155, and 178.

Call Format

```
CALL   'TRS'   'DYP'  parameters
```

Required Parameters	Format	Length	In/Output
Return Code	binary	4 bytes	output
Dynamic Parameters	alphanumeric	variable	input

Return Code

The return code is the message delivered at the end of processing which indicates whether an error has occurred. A zero code indicates the normal end of processing. Other codes are explained in Appendix A **Messages and Codes**.

Dynamic Parameters

Any of the following dynamic parameters can be specified in a DYP call:

AUTOASP	**DSFNR**	**INCVOC**	**PASSWORD**
CONCHAR	**ERRADA**	**INDEX**	**SEARCHLB**
DBTYPE	**ERRPRE**	**TEXT**	**SETCHAR**
DEFOPER	**ERRUSE**	**MAXDPRO**	**TRUNCHAR**
DFNR	**HIGHLIGHT**	**MAXVSET**	**VFNR**
DOCID	**HOLDWORD**	**MULTICALL**	**WORDLEN**

For details of the dynamic parameters and their possible values, refer to the section entitled **Dynamic Parameters for the DYP and BC Calls** on page 40.

EISE

Description

Each query executed by Adabas Text Retrieval results in an Adabas ISN set. This set can be referenced by EISE, EISG and EISS calls. After an EISS call and any number of EISG calls have been used, the EISE call must be used to conclude browsing through an ISN set.

Example

See page 173.

Call Format

```
CALL 'TRS' 'EISE' parameters
```

Required Parameters	Format	Length	In/Output
Return Code	binary	4 bytes	output
Command-ID	binary	4 bytes	input
Set Type	alphanumeric	1 byte	input

Return Code

The return code is the message delivered at the end of processing which indicates whether an error has occurred. A zero code indicates the normal end of processing. Other codes are explained in Appendix A **Messages and Codes**.

Command-ID

The Adabas Command-ID of the ISN set to be referenced, as created by the QR call. It is identical to the *Command-ID* output parameter of the QR call which generated the ISN set and must therefore be set to the same value.

Set Type

The type of ISN set for which browsing is to be terminated. One of the following values must be specified:

'D'	Document ISN set
'V'	Vocabulary ISN set

The value of the *Set Type* parameter must be identical to the value of the *Type* parameter of the QR call.

EISG

Description

Each query executed by Adabas Text Retrieval results in an Adabas ISN set. This set can be referenced by EISE, EISG and EISS calls.

The EISG call is used to browse through an ISN set created by a QR call. The sequence of one or more EISG calls must be preceded by an EISS call and concluded by an EISE call.

Example

See page 170.

Call Format

CALL ’TRS’ ’EISG’ *parameters*

Required Parameters	**Format**	**Length**	**In/Output**
Return Code	binary	4 bytes	output
Command-ID	binary	4 bytes	input
Set Type	alphanumeric	1 byte	input
Quantity	binary	4 bytes	input
Position	binary	4 bytes	input
ISN	binary	4 bytes	output

Return Code

The return code is the message delivered at the end of processing which indicates whether an error has occurred. A zero code indicates the normal end of processing. Other codes are explained in Appendix A **Messages and Codes**.

Command-ID

The Adabas Command-ID of the ISN set to be referenced, as created by the QR call. It is identical to the *Command-ID* output parameter of the QR call which generated the ISN set and must therefore be set to the same value.

Set Type

The type of ISN set for which browsing is to be executed. One of the following values must be specified:

'D'	Document ISN set
'V'	Vocabulary ISN set

The value of the *Set Type* parameter must be identical to the value of the *Type* parameter of the QR call.

Quantity

The number of ISNs in the set generated by the QR call. The value of the *Quantity* parameter must be identical to the *Quantity* parameter of the QR call.

Position

The position of the requested *ISN* within the ISN set as generated by the QR call.

ISN

The ISN within the position as indicated by the *Position* parameter in the ISN set is returned by the EISG call.

EISS

Description

Each query executed by Adabas Text Retrieval results in an Adabas ISN set. This set can be referenced by EISE, EISG and EISS calls.

The EISS call starts browsing through an ISN set created by a QR call. The EISS call must be performed once before each sequence of EISG calls used to browse through an ISN set.

Example

See page 169.

Call Format

```
CALL 'TRS' 'EISS' parameters
```

Required Parameters	Format	Length	In/Output
Return Code	binary	4 bytes	output
Command-ID	binary	4 bytes	input
Set Type	alphanumeric	1 byte	input
Quantity	binary	4 bytes	input

Return Code

The return code is the message delivered at the end of processing which indicates whether an error has occurred. A zero code indicates the normal end of processing. Other codes are explained in Appendix A **Messages and Codes**.

Command-ID

The Adabas Command-ID of the ISN set to be referenced, as created by the QR call. It is identical to the *Command-ID* output parameter of the QR call which generated the ISN set and must therefore be set to the same value.

Set Type

This parameter defines the type of ISN set for which browsing is to be started. One of the following values must be specified:

’D’	Document ISN set
’V’	Vocabulary ISN set

The value of the *Set Type* parameter must be identical to the value of the *Type* parameter of the QR call.

Quantity

The number of ISNs in the set generated by the QR call. The value of the *Quantity* parameter must be identical to the *Quantity* parameter of the QR call.

HIGH

Description

The HIGH call is used to mark those words in a document which have been found for a given query.

The HIGH call marks the beginning and ending of the words to be highlighted by two specific characters. In NATURAL these characters can be used for dynamic highlighting by means of the dynamic attribute feature (DY).

Important

Before the HIGH call can be executed, the free-text chapter to which the entered text belongs must have been established.

The free-text chapter is established by a BC or DYP call which provides the name of the Adabas hyperdescriptor associated with the free-text chapter in question, as the value of the TEXT parameter.

Example

See page 177.

Call Format

```
CALL  'TRS'  'HIGH'  parameters
```

Required Parameters	Format	Length	In/Output
Return Code	binary	4 bytes	output
Document-ID	alphanumeric	variable	input
Query Name	alphanumeric	8 bytes	input
Input Text	alphanumeric	variable	input
Output Text	alphanumeric	variable	output
Text Length	binary	4 byte	input
Prefix	alphanumeric	1 byte	input
Suffix	alphanumeric	1 byte	input
Cursor	binary	4 bytes	in-/output

Return Code

The return code is the message delivered at the end of processing which indicates whether an error has occurred. A zero code indicates the normal end of processing. Other codes are explained in Appendix A **Messages and Codes**.

Document-ID

A unique Document-ID must be provided. There are two different ways of providing a Document-ID depending on the setting of the DOCID parameter of the DYP call:

- If the DOCID parameter (DYP call) contains the name of a formatted field (Adabas descriptor), the *Document-ID* parameter must contain a unique value for the formatted field in question.
- If the DOCID parameter (DYP call) contains the value "**##**", the *Document-ID* parameter must contain the Adabas ISN of a record on the document file representing the document in question.

Query Name

The name of the query as defined in the *Query Name* parameter of the QR call. The words will be highlighted according to the selection criteria specified in this query.

Input Text

The source text which contains the words to be highlighted. Blanks should be provided at the beginning and at the end of the source text to host the assigned prefix and suffix characters if necessary.

Output Text

The source text including the assigned prefix and suffix characters.

Text Length

The length of the source text expressed in bytes.

Prefix

The special character indicating the beginning of a word to be highlighted.

Recommended character: “**<**”

Suffix

The special character indicating the end of a word to be highlighted.

Recommended character: “**>**”

Cursor

For each free-text chapter of a document, this parameter has to be set to zero at the beginning of the highlighting process and must not be changed for the remainder of the process.

PHON

Description

The PHON call is used to translate a token into a phonetic value.

Call Format

```
CALL  'TRS'  'PHON'  parameters
```

Required Parameters	Format	Length	In/Output
Return Code	binary	4 bytes	output
Truncation Character	alphanumeric	1 byte	input
Token to be Translated	alphanumeric	variable (1-64 bytes)	input
Translated Value	alphanumeric	64 bytes	output
Phonetic ID	alphanumeric	1 byte	input
Length of Token	binary	4 bytes	input
Number of Phon. Values	binary	4 bytes	output

Return Code

The return code is the message delivered at the end of processing which indicates whether an error has occurred. A zero code indicates the normal end of processing. Other codes are explained in Appendix A **Messages and Codes**.

Truncation Character

Specifies the character to be used as word-truncation character. This should be the same character as specified by the BC or DYP call.

Token to be Translated

Specifies the token to be translated.

Translated Value

The resulting translated token.

Phonetic ID

Not used at the moment but must be specified.

Length of Token

The length of the token to be translated in bytes.

Number of Phon. Values

The number of resulting tokens after translation. In the case of an error, the returned value is 0 (zero).

QR

Description

The QR call is used to retrieve text and information from free-text chapters and formatted fields. This process is called information retrieval.

Example

See pages 160, 162, 163, 169, 183, 189.

Call Format

```
CALL 'TRS' 'QR' parameters
```

Required Parameters	Format	Length	In/Output
Return Code	binary	4 bytes	output
Query	alphanumeric	variable	input
Query Length	binary	4 byte	input
Query Name	alphanumeric	8 bytes	input
Disp Error	binary	4 byte	output
Length Error	binary	4 byte	output
Default Mode	alphanumeric	1 byte	input
Command-ID	binary	4 bytes	output
Quantity	binary	4 bytes	output
Type	alphanumeric	1 byte	input

Return Code

The return code is the message delivered at the end of processing which indicates whether an error has occurred. A zero code indicates the normal end of processing. Other codes are explained in Appendix A **Messages and Codes**.

Query

The query. Its syntax is described in the Chapter **Query Syntax**.

Query Length

The length of the current query expressed in bytes.

Query Name

The name of the current query. Subsequent queries can use this name to refer to the results of the current query. Two different types of query names are possible depending on the value of the *Type* parameter:

Type	Query Name	Range of *nnn*
D	DOCS0nnn	001–999
V	WRDS0nnn	001–999

Disp Error

If an error is detected when the syntax of the query is checked, this parameter will contain the displacement of the erroneous term within the query.

Length Error

If an error is detected when the syntax of the query is checked, this parameter will contain the length of the erroneous term within the query.

Default Mode

The default selection mode. One of the following letters must be specified as the default selection mode:

Letter	Selection Mode
=	PRECISE
A	ASPECT
G	GROUP
P	PHONETIC
R	ROOT
S	SYN
X	SYR

Selection modes are explained in Chapter **Query Syntax**.

Command-ID

The Adabas Command-ID of the Adabas ISN set created by the QR call.

This parameter serves as input to the EISS, EISG, EISE and RET calls.

Quantity

The number of ISNs contained in the ISN set created by the Adabas Text Retrieval QR call.

Type

The type of retrieval to be executed by the QR call. There are two possible values:

‘D‘	Document retrieval
‘V‘	Vocabulary retrieval

RQR

Description

The RQR call is used to release all results of a specific query.

A query is automatically released when another QR call with the same Query Name is executed.

Example

See pages 159, 162, 163, 188.

Call Format

```
CALL 'TRS' 'RQR' parameters
```

Required Parameters	Format	Length	In/Output
Return Code	binary	4 bytes	output
Query Name	alphanumeric	8 bytes	input

Return Code

The return code is the message delivered at the end of processing which indicates whether an error has occurred. A zero code indicates the normal end of processing. Other codes are explained in Appendix A **Messages and Codes**.

Query Name

The name of the query to be released, as assigned in the QR call.

RULE

Description

The RULE call is used to define the rules for document inversion.

Call Format

```
CALL 'TRS' 'RULE' parameters
```

Required Parameters	Format	Length	In/Output
Return Code	binary	4 bytes	output
Option	alphanumeric	7 bytes	input
Max Words	binary	4 bytes	input
Aspects	alphanumeric	variable	input
Marks	alphanumeric	variable	input

Return Code

The return code is the message delivered at the end of processing which indicates whether an error has occurred. A zero code indicates the normal end of processing. Other codes are explained in Appendix A **Messages and Codes**.

Option

This parameter defines the type of inversion to be executed. The options are:

Option	Characteristics
’FULL’	Full text inversion of all words contained in the document to be inverted. This option is generally the default setting.
’EXCLUDE’	Inversion of all words not belonging to one of the aspects as defined in the *Aspect* parameter.
’INCLUDE’	Inversion only of those words belonging to aspects as defined in the *Aspect* parameter.
’MARKED’	Inversion only of those words which are tagged with special markers as defined in the *Marks* parameter.

Max Words

The maximum number of words to be inverted per document. This parameter is only maintained for upward compatibility. If Adabas hyperdescriptors are used for document indexing then this parameter is of no relevance.

Aspects

A list of the aspects to be used for the inversion of a document. Individual aspects on the list must be separated by commas and the list must end with a period.

Marks

A list of predefined character strings used in the document to be inverted. If the mark used refers to a predefined aspect, the latter can be entered after the mark and separated from it by the equals (=) symbol. Marks must be separated by commas and the list concluded with a period.

Dynamic Parameters for the DYP and BC Calls

The following dynamic parameters can be specified by a BC or DYP call.

Parameter	Explanation
AUTOASP	Automatically creates an aspect for each word entered in the vocabulary file.
CONCHAR	Specifies a character in hexadecimal format to be used to concatenate multi-word terms. Blank and binary zero must not be specified. The format is: *hh* Example: **'CONCHAR=2D'**
DBTYPE	Specifies the target (server) environment. UNIX is the default environment. Possible values are: UNIX, VMS, MAINFRAME. Example: **'DBTYPE=VMS'**
DEFOPER	Specifies the default operator to be used in case none is explicitly mentioned between two terms. Possible values are: AND, OR, NOT, ADJ, INPAR, INSEN, NEAR Default setting: ADJ
DFNR	Document File Number. The Document File Number can be specified either as it stands, or alternatively together with the Adabas Database ID in question. In the latter case, Database ID and file number must be enclosed in parentheses and separated by a comma. Example: **'DFNR=39'** **'DFNR=(1,39)'** *Note: The DFNR parameter is mandatory*
DOCID	Adabas field name of the Document-ID in the document file. The value **'##'** indicates that the ISN of the document file is to be used as document identification. Default setting: **'DA'**

Parameter	Explanation
DSFNR	Document Index File Number. The Document Index File Number can be specified either as it stands, or alternatively together with the Adabas Database ID in question. In the latter case, Database ID and file number must be enclosed in parentheses and separated by a comma. Example: **'DSFNR=39'** **'DSNR=(1,39)'** *Note: The DSFNR parameter is mandatory*
ERRADA	A constant value to be added to all Adabas return codes. You may specify 5 digits. Default setting: 0 (zero)
ERRPRE	A constant value to be added to all Adabas Text Retrieval return codes. You may specify 5 digits. Default setting: 0 (zero)
ERRUSE	A constant value to be added to all User Exit return codes. You may specify 5 digits. Default setting: 0 (zero)
HIGHLIGHT	Specifies the highlight algorithm. Possible values are: R, I R = Replace the first blank found below/after the word to be highlighted with the highlight characters. This is the default. I = Insert before and after the word to be highlighted the highlight characters. Because of character insertion, the output buffer may be larger after return. Check buffer length after return. The output buffer should be large enough to hold the complete text plus highlight characters. Example: **'HIGHLIGHT=R'**
HOLDWORD	Possible values are: YES, NO To ensure correct document inversion in multi-user environments, this parameter should be set to YES. If YES, each word is read in HOLD. Default setting: NO Example: **'HOLDWORD=YES'**

Parameter	Explanation
INCVOC	The format is: *aaaaaaaa* Specifies the name (maximum 8 bytes) of incore vocabulary. For more information, refer to Chapter **Incore Vocabulary**. Example: **'INCVOC=MYWORDS'**
INDEX	This parameter indicates which proximity indices are to be maintained by Adabas Text Retrieval. The following values can be entered: **'INDEX=(WORD)'** only word positions are maintained. **'INDEX=(WORD,SENTENCE)'** word and sentence positions are maintained. **'INDEX=(WORD,PARAGRAPH)'** word and paragraph positions are maintained. If this parameter is omitted, Adabas Text Retrieval will maintain word, sentence and paragraph positions.
MAXDPRO	Specifies the maximum number of selected documents on which a proximity search is to be performed. Default setting: 200
MAXVSET	Specifies the maximum number of words for a search. Default setting: 2000
MULTICALL	Possible values are: YES, NO Specifies whether the /Adabas multicall feature is to be used during the document inversion process. If the multicall feature is used, the Adabas Text Retrieval buffer size should be at least 96 KB. Default setting: NO The following parameter settings apply: **–** MULTICALL=YES activates the Multi-Call command feature. **–** MULTICALL=NO deactivates the Multi-Call command feature.

Parameter	Explanation
	Using the feature leads to significant increase in performance during document inversion, in particular in client/server applications. Example: **'MULTICALL=YES'**
PASSWORD	An Adabas password can be supplied.
SEARCHLB	Specifies the default search label. The names of free-text chapters can be entered to a maximum of 20. If this parameter is omitted it takes the current value of the TEXT parameter. Example: **'SEARCHLB=(Y1Y1,Y2Y2,Y3Y3)'**
SETCHAR	Specifies the prefix character used to identify the result of previous queries. Default setting: NONE Recommended setting: **#**
TEXT	The name of the Adabas hyperdescriptor of the current free-text chapter. This name is used for subsequent ADD and DDS calls. For compatibility reasons regarding previous versions of Adabas Text Retrieval, the name of the hyperdescriptor must be entered twice. If the SEARCHLB parameter is omitted, the value of the TEXT parameter is used as the default search label. Default setting: **'Y1Y1'**
TRUNCHAR	The character to be used as word truncation indicator. Default setting: **'*'**
VFNR	Vocabulary File Number. The Vocabulary File Number can be specified either as it stands, or alternatively together with the Adabas Database ID in question. In the latter case, Database ID and file number must be enclosed in parentheses and separated by a comma. Example: **'VFNR=38'** **'VFNR=(1,38)'** *Note: The VFNR parameter is mandatory*

WORDLEN	The value of this parameter indicates the word length to be used by Adabas Text Retrieval. Maximum word length: 64. Default setting: 32

The dynamic parameters form a character string of one or more parameter entries and must be separated by commas and ended by a period. Each parameter must be coded as follows:

’Parameter=Value’

Example:

’VFNR=38,DFNR=39,DSFNR=39,TEXT=Y2Y2.’

QUERY SYNTAX

Search Labels

A search label is an alphanumeric identifier up to eight characters long used to refer to an element name in a search query. For example, the label “ABS” could be used to refer to the element “ABSTRACT” in search queries. This method of abbreviation saves keystrokes in entering search queries.

A global search label is a search label that represents a combination of two or more search labels. Several labels can be combined to form a global label which, when applied in search queries, can be used to search for more than one text inverted element. For example, the search labels ABSTRACT and TITLE could be assigned the global label ABSTI.

Search labels are defined using the DSL call (see page 19).

Queries in formatted fields must be preceded by a search label.

Queries to free-text chapters can be preceded by a search label, but do not have to be if the search label required is the same as that last specified in the TEXT or SEARCHLB parameter of the last BC or DYP call, so-called standard search labels. Thus, a search label for a free-text chapter remains valid until another search label has been chosen to replace it. For example:

```
ABS ADABAS
```

The occurrences of the term ADABAS within the free-text chapter ABSTRACT will be retrieved.

Search Mode Parameters

The search mode indicates the method to be used when retrieving inverted words.

The search mode precedes the search term in a query. If no search mode is specified, then the mode most previously specified is used. The following syntax must be used:

```
    search-label   search-mode   search-term
```

When no selection mode is specified in the QR call which initiates the query, the default selection mode is used. The default selection mode is defined by the value of the Default Selection Mode parameter of the QR call in question.

search-label	The name of the search label for the element concerned
search-term	The term to be used as the basis for retrieval
search-mode	One of the following parameters

Parameter*	**Mode**
=	PRECISE
PHONETIC	PHONETIC
SYN	SYNONYM
ROOT	ROOT
SYR	SYNONYM/ROOT
ASPECT	ASPECT
GROUP*n*	GROUP

The options are explained in more detail below.

PRECISE Mode

The default mode. Searches are performed on the basis of spelling alone. Input must be identical to that contained in the document.

PHONETIC Mode

All words are retrieved which have the same phonetic value. This feature is designed for searches in German language, but it can also be used with some success for English language. For example, a PHONETIC search for the name “Mayer” will also retrieve the names “Maier” and “Meyer”.

SYNONYM Mode

If search terms are have synonyms defined for them in a thesaurus, then all documents containing the search term and its synonyms are found. The selection mode SYNONYM is based on the information stored previously in the SYNONYM field (V8) of the vocabulary file.

ROOT Mode

The ROOT search mode selects those documents which contain any of the search term's previously defined roots. This search mode is based on the information stored previously in the ROOT field (V4) of the vocabulary file.

SYNONYM/ROOT Mode

The SYR selection mode selects those documents which contain the words specified in the query, and/or any of their previously defined synonyms, and/or any of their previously defined roots. This search mode is based on the information stored previously in the SYNONYM field (V8) and/or ROOT field (V4) of the vocabulary file.

ASPECT Mode

All words are retrieved which are narrower terms for the search term. The number of hierarchical levels between the search term and the terms found is irrelevant. For example, an ASPECT search for the term "fiction" would retrieve the words "poem", "epic", "sonnet", "ballad", "haiku", "novel", "story", etc. The selection mode ASPECT is based on the information stored previously in the ASPECT field (V5) of the vocabulary file.

GROUP Mode

All words are retrieved which are narrower terms occurring *n* levels lower in the thesaurus hierarchy than the search term. Where *n* is omitted the depth is equal to 1.

For example, a GROUP search for the term "fiction" would retrieve the words "poem", "novel", "story", etc., but NOT the narrower terms ("sonnet", "epic", "ballad", "haiku", etc.) for these.

The selection mode GROUP is based on the information stored previously in the ASPECT field (V5) of the vocabulary file.

Query Syntax Diagrams

The syntax diagram below shows the various elements of a search query. Expressions in square brackets are optional.

QUERY= *query-expression* [*boolean-operator query*] [**SORT** | **SORTD** *sort-field*]

The content of *query-expression* varies depending on whether you reference word-inverted elements, formatted-inverted elements, or search numbers. The different possibilities are listed below.

Word-inverted Elements (free-text chapters)

QUERY EXPRESSION = *search-label word-set* [*proximity-operator word-set*]

Formatted-inverted Fields

QUERY EXPRESSION = *search-label* [*relational-operator*] *word-set*

Search Numbers

QUERY EXPRESSION = *search-number* [*boolean-operator query*]

Search Query Language

Case

Search queries may be entered in either upper or lower case.

Blanks

In searching formatted-inverted text elements, search terms which contain blanks or non-TRS characters must be enclosed in quotes.

In searching word-inverted text elements, blanks between words are interpreted as the value set in the parameter DEFOPER.

Reserved Words

Search labels, global labels, and operators are reserved words within TRS and therefore cannot be used directly in search queries. To apply a reserved word in a search query, it must be surrounded by quotes.

Truncation

Truncation enables the retrieval of documents containing word segments or derivatives. Three types of truncation are available: left, right and middle. The character used to indicate truncation is specified by your administrator during installation. The following examples illustrate the three truncation options applying an asterisk (“*”) as truncation character.

Note:
*Only **right** word truncation is possible for formatted-inverted fields.*

Right Truncation

Right truncation is used to retrieve documents which contain words beginning with a specified string of letters. Thus the query

```
ABSTRACT  kilo*
```

retrieves all words beginning with the string “kilo”, such as “kilogram” and “kilowatt”.

Left Truncation

Left truncation is used to retrieve documents which contain words ending with a specified string of letters. Thus the query

ABSTRACT *gram

retrieves all words ending with the string “gram”, such as “kilogram” and “program”.

Note:
Not possible for formatted-inverted fields.

Left and Right Truncation

Combined right and left truncation is used to retrieve documents which have words containing a specified string of letters. Thus the query

ABSTRACT *ra*

retrieves words such as “gram”, “kilogram”, “hurrah”, “arab”, etc.

Note:
Not possible for formatted-inverted fields.

Middle Truncation

Middle truncation is used to retrieve documents which contain words beginning and ending with a specified string of letters. Thus the query

ABSTRACT hypo*mia

retrieves all words beginning with the string “hypo” and ending with the string “mia”, such as “hypothermia” and “hypoglycaemia”. It is not possible to specify how many letters are to be truncated.

Note:
Not possible for formatted-inverted fields.

Operators

Evaluation Order of Operators

When several different operators are used in the same search query, the order of evaluation is determined on a the basis of predefined priority. The evaluation priorities, from highest to lowest, are the following:

1. Expressions enclosed in parentheses;
2. AND;
3. NOT;
4. OR.

Boolean Operators

Boolean operators are used to join query expressions of the same or different types. The following Boolean operators may be used:

- AND ;
- NOT;
- OR.

The AND Operator

The AND operator is used to select documents based on the commonality of two query expressions. For example, the query

ABSTRACT strawberries AND cream

retrieves all documents in which the words "strawberries" and "cream" both occur in the element with the ABSTRACT search label.

The AND operator can be used more than once in a single query. For example, the query

ABSTRACT gin AND vermouth AND olive

retrieves all documents in which all the words are contained.

The OR Operator

The OR operator is used to retrieve documents in which any one of the terms specified occur. This is especially useful for retrieving related concepts. For example, the query

```
ABSTRACT sugar OR sweetener
```

retrieves all documents in which either the word “sugar” or the word “sweetener” occurs in the element with the ABSTRACT search label.

Like the AND operator, the OR operator can be used more than once within a single query. You can also replace the OR in search queries with a comma.

```
ABSTRACT sugar, sweetener
```

The NOT Operator

The NOT operator is used to retrieve documents which contain one specified term and which do not contain another. It can be used only following a query expression. For example, the query

```
ABSTRACT sweetener NOT honey
```

retrieves those documents in which the element with the ABSTRACT search label contains the word “sweetener”, but not the word “honey”. The following example, however, is invalid:

```
ABSTRACT NOT honey
```

Relational Operators

Relational operators are used to reference alphanumeric and numeric formatted-inverted elements.

The following relational operators may be used:

- BETWEEN *n*,*n*
- EQ *n* ("equal to")
- GE *n* ("greater than or equal to")
- GT *n* ("greater than")
- LE *n* ("less than or equal to")
- LT *n* ("less than")

where "*n*" in each case is an obligatory value which depends on the element in question.

The relational operator can be omitted in which case the default is 'EQ'.

The operators are described in more detail below.

The BETWEEN Operator

The BETWEEN operator is used to retrieve documents containing any one of a range of values. For example, the query

DATE BETWEEN 19870101,19871231

retrieves all documents in which the date given in the element with the DATE search label is 1987.

The EQ Operator

The EQ ("equal to") operator is used to retrieve documents containing a single, precise value. For example, the query

NUMBER EQ 109

retrieves the document in which the value for the NUMBER search label is 109.

The same result is also be achieved by omitting the operator:

NUMBER 109

The GE Operator

The GE (“greater than or equal to”) operator is used to retrieve documents containing a value greater than or equal to the specified value. For example, the query

NUMBER GE 109

retrieves all documents in which the value for the NUMBER search label is 109 or greater.

The GT Operator

The GT (“greater than”) operator is used to retrieve documents containing a value larger than the specified value. For example, the query

NUMBER GT 109

retrieves all documents in which the value for the NUMBER search label is greater than 109.

The LE Operator

The LE (“less than or equal to”) operator is used to retrieve documents containing a value less than or equal to the specified value. For example, the query

NUMBER LE 109

retrieves all documents in which the value for the NUMBER search label is less than or equal to 109.

The LT Operator

The LT (“less than”) operator is used to retrieve documents containing a value smaller than the one specified. For example, the query

NUMBER LT 109

retrieves all documents in which the NUMBER search label is less than 109.

Proximity Operators

Proximity operators specify retrieval based on relative word position within a text. They can only be used to search word-inverted elements.

The following proximity operators are available:

- ADJ
- NEAR

The ADJ Operator

The ADJ operator is used to retrieve documents in which words appear next to one another and in the order specified. For example, the query

ABSTRACT Moon ADJ River

selects all documents in which the element with the “ABSTRACT” search label contains the word string “Moon River”.

The NEAR Operator

The NEAR operator is used to retrieve documents in which words appear next to each other, irrespective of their order. For example, the query

ABSTRACT recycled NEAR paper

selects all documents in which the element with the ABSTRACT search label contains the word pairs “recycled paper” and “paper recycled”.

Search Numbers

Search numbers are the numbers allocated to each query you issue during a session. They can be used in subsequent queries to reference the set of documents already retrieved. To distinguish them from normal numbers in queries, they must be given a prefix (e.g. #1). The prefix is specified by your administrator during installation.

The search number consists of the current value of the SETCHAR parameter and the number of the query whose results are to be referenced.

An example of a query using a search number is:

```
#1 AND AUTHOR DICKENS
```

This would retrieve all documents in query set 1 with the author Dickens.

The SORT and SORTD Functions

Using the SORT and SORTD functions, you can sort a document set in ascending or descending order.

The syntax of the sort function is as follows:

```
SORT search-label
```

```
SORTD search-label
```

SORT	Specifies a sort in order of ascending magnitude.
SORTD	Specifies a sort in order of descending magnitude
search-label	Specifies the formatted-inverted element to be used as the sort criterion.

For example, the query

ABSTRACT sugar SORT DATE

retrieves all documents in which the word “sugar” occurs in the element with the ABSTRACT search label and sorts them according to the value given in the element which has search label “DATE”, starting with the oldest document.

As many as three sort criteria can be specified in a search query, for example:

SORT NUMBER SORT PUBLISH SORT ACC-NO

TOKENIZATION

Tokenization is the process by which Adabas Text Retrieval identifies words in text.

The tokenization process has been implemented as a Adabas Text Retrieval routine, callable during the initiation of the system. It is possible to save the results for future use. Adabas Text Retrieval constructs several tables which are used in the tokenization process.

The process consists of the following parts:

- Define classes of characters to be used by Adabas Text Retrieval (for example, alphanumeric, numeric).
- Identify of valid characters. The Adabas Text Retrieval character table contains valid characters.
- Determine whether tokenization is dependent on the context of the characters.
- Translate the word detected by the previous parts of the process.

The entire tokenization process is optional.

Associated with each Adabas Text Retrieval call is a return code. It indicates whether an error has occurred and, if so, the type of error. A list of possible return codes is provided in section "Return Codes" in **Appendix A**, Messages and Codes.

Specified in each Adabas Text Retrieval call is a scan mode, which determines whether the scan is to be performed for text inversion or query-syntax analysis, or for both. A list of possible scan modes is provided in section "Scan Modes" on page 60.

During the text-inversion process or query-syntax analysis, Adabas Text Retrieval looks for a user-defined scan logic. If user-defined scan logic does not exist, Adabas Text Retrieval uses its default scan logic.

Scan Modes

The scan mode determines whether the scan is to be performed for text inversion or query-syntax analysis, or for both. The table below contains the letter codes for the scan modes:

A	Both text inversion process and query syntax analysis.
F	On user request only (for example, translation of formatted fields value).
P	Parameter syntax analysis only.
Q	Query-syntax analysis only.
T	Text inversion process only.

SCTS – Define Classes of Characters

Description

The SCTS call defines all classes of characters to be used by Adabas Text Retrieval. It is always the first call in the tokenization process. Typical character classes are ALPH, NUMBER, DOT, etc.

Call Format

```
Call  'TRS'  'SCTS'  parameters
```

Required Parameters	Format	Length	In/Output
Return Code	binary	4 bytes	output
Scan Mode	alphanumeric	1 byte	input
Classes	alphanumeric	variable	input

Classes

A list of all classes of characters which will be used by Adabas Text Retrieval. Classes on the list must be separated by commas; the last class must be followed by a period.

The maximum number of classes is 16; the maximum length of each class is six characters.

Example SCTS Call

```
CALL 'TRS' 'SCTS' RETURN–CODE 'T' 'ALPH,NUMBER,DOT,BLANK,SEP.'
```

This definition is for a text-inversion process only ('T') and the user-defined classes of characters to be used by Adabas Text Retrieval are ALPH, NUMBER, DOT, BLANK and SEP.

SCTC – Define Character Table

Description

The SCTC call is used to assign specific characters (a character table) to one or more of the classes already defined by the SCTS call (see page 61).

Note:
If you do not supply Adabas Text Retrieval with a character table, it uses a default table. In this case, make sure that the default table is compatible with its definition of classes.

Call Format

```
Call   'TRS'   'SCTC'   parameters
```

Required Parameters	Format	Length	In/Output
Return Code	binary	4 bytes	output
Scan Mode	alphanumeric	1 byte	input
Character Table	alphanumeric	variable	input
Line Error	binary	4 bytes	output

Character Table

A character table is a set of lines which assign specific characters to one or more classes. All lines are fixed length, 80 characters long.

The format of each line is:

CHAR=(c1,c2,c3...), CLASS=class1

where *c1,c2,c3*... denotes the characters to be assigned to the character class *class1*. The characters can be entered in either character format or hexadecimal format. Input in hexadecimal format must start with the letter ‘X’ followed by a two-byte hexadecimal value (total of three bytes). Input is in character format is one byte in length. All values must be separated by commas and enclosed by parentheses. It is not necessary to define all of the 256 possible characters. Those which are not defined are regarded as delimiters. It is possible to assign characters to more than one class.

The classes specified must be already defined in the SCTS call (see page 61). The set can contain as many lines as required to define all relevant characters.

The set is ended by a line containing the constant “END”.

Line Error

If an error is detected when the syntax of the character table is checked, this parameter contains the number of the line containing the error.

Example SCTC Call

The following is an example of the SCTC call:

```
CALL ’TRS’ ’SCTC’ RETURN–CODE ’A’ CHAR–TABLE LINE–ERROR
```

where **CHAR–TABLE** is a pointer to:

```
CHAR=(A,B,C,D,E,F,G,H,I,J),CLASS=ALPH
CHAR=(K,L,M,N,O,P,Q,R,S,T),CLASS=ALPH
CHAR=(U,V,W,X,Y,Z),CLASS=ALPH
CHAR=(a,b,c,d,e,f,g,h,i,j),CLASS=ALPH
CHAR=(k,l,m,n,o,p,q,r,s,t),CLASS=ALPH
CHAR=(u,v,w,x,y,z),CLASS=ALPH
CHAR=(1,2,3,4,5,6,7,8,9,0),CLASS=NUMBER
CHAR=(–),CLASS=SEP
CHAR=(!,?,.),CLASS=DOT
CHAR=(X20),CLASS=BLANK
END
```

SCTT – Define Translation Table

Description

The SCTT call defines a translation table. This translation table is used by Adabas Text Retrieval to translate a token after it is isolated.

Note:
The default translation is no translation at all.

Call Format

```
Call 'TRS' 'SCTT' parameters
```

Required Parameters	Format	Length	In/Output
Return Code	binary	4 bytes	output
Scan Mode	alphanumeric	1 byte	input
Translation Code	alphanumeric	5 bytes	input
Translation Table	alphanumeric/binary	variable/256 bytes	input
Line Error	binary	4 bytes	output

Translation Code

The translation code specifies the type of translation:

ASIS	No translation at all. All characters remain as they were given by the application.
CHAR	Characters are translated according to a user-defined translation table.
UPPER	All characters are translated to upper case.
LOWER	All characters are translated to lower case.
TABLE	All characters are translated according to a user-defined translation table.

Translation Table

The contents of the translation table depend on the translation code. For translation codes ASIS, UPPER and LOWER, no translation table can be specified.

For translation code TABLE, a 256-byte table defining the translated character value for all 256 characters must be specified.

For translation code CHAR, a translation table with variable length must be specified. The table contains only those characters which are to be translated. The syntax to define the translation table is as follows:

LC=*char*,UC=*transchar*,LC=*char*,UC=*transchar*[,...]
LC=X*hex*,UC=X*transhex*,LC=X*hex*,UC=X*transhex*[,...]
...
END

where *char* is a character in ASCII format, *transchar* is the translated character value in ASCII format, *hex* is a character in hexadecimal representation and *transhex* is the translated character value in hexadecimal representation.

Example

```
LC=a,UC=A,LC=b,UC=B
LC=X61,UC=X41,LC=X62,UC=X42
END
```

Line Error

Contains the number of the line containing an error.

Example SCTT Call

The following is an example of the SCTT call. It demonstrates a simple translation from lower case to upper-case:

```
CALL 'TRS' 'SCTT' TRS.RC 'T' 'TABLE' #SCAN-TABLE(*)
```

SCTW – Define Reserved Words

Description

Defines reserved words to the Adabas Text Retrieval keyword table.

Call Format

```
Call 'TRS' 'SCTW' parameters
```

Required Parameters	Format	Length	In/Output
Return Code	binary	4 bytes	output
Keywords	alphanumeric	variable	input
Line Error	binary	4 bytes	output

Keywords

A set of statements which define the reserved words. The table of reserved words has the following syntax:

class=(*reservedword*, *reservedword*[,...])[,DELETE]
...
END

where *class* is one of the following identifiers:

- STDFUNCTION
- COMMA
- OR
- AND
- NOT

- EQUALTO
- GREATERTHAN
- GREATEREQUAL
- LESSTHAN
- LESSEQUAL
- NEAR
- GROUP
- ADJACENT
- BETWEEN
- SYNONYM
- SYNONYMROOT
- SYNONYMNEW
- SORTASCENDING
- SORTDESCENDING
- ROOT
- PHONETIC
- CATEGORY
- ASPECTNARROW
- ASPECTBROADER
- LEFTPARENTHESES
- RIGHTPARENTHESES
- INSENTENCE
- INPARAGRAPH
- USERFUNCTION0
- USERFUNCTION1
- USERFUNCTION2
- USERFUNCTION3

- USERFUNCTION4
- USERFUNCTION5
- USERFUNCTION6
- USERFUNCTION7
- USERFUNCTION8
- USERFUNCTION9

The DELETE parameter is optional and, if specified, removes all previously defined, reserved words of the given class. The following table shows the default definitions:

- StdFunction = (=)
- Comma = (,)
- Or = (OR)
- And = (AND)
- Not = (NOT)
- EqualTo = (EQ)
- GreaterThan = (GT)
- GreaterEqual = (GE)
- LessThan = (LT)
- LessEqual = (LE)
- Near = (NEAR)
- Group = (GROUP)
- Adjacent = (ADJ)
- Between = (BETWEEN)
- Synonym = (SYN)
- SynonymRoot = (SYR)
- SynonymNew = (NSYN)
- SortAscending = (SORT)
- SortDescending = (SORTD)

- Root = (ROOT)
- Phonetic = (PHONETIC)
- Category = (ASPECT,CATEGORY)
- AspectNarrow = (NASPECT)
- AspectBroader = (TBT)
- LeftParentheses = (()
- RightParentheses = ())
- InSentence = (.,!,?)
- InParagraph = ($$)
- UserFunction0 = (USRFUNC0)
- UserFunction1 = (USRFUNC1)
- UserFunction2 = (USRFUNC2)
- UserFunction3 = (USRFUNC3)
- UserFunction4 = (USRFUNC4)
- UserFunction5 = (USRFUNC5)
- UserFunction6 = (USRFUNC6)
- UserFunction7 = (USRFUNC7)
- UserFunction8 = (USRFUNC8)
- UserFunction9 = (USRFUNC9)

Line Error

Contains the number of the line containing an error.

SCTX – The Tokenization Logic

Description

The SCTX call defines actions to be taken by Adabas Text Retrieval when the tokenization process identifies a character belonging to a specific class.

Call Format

```
Call  'TRS'  'SCTX'  parameters
```

Required Parameters	Format	Length	In/Output
Return Code	binary	4 bytes	output
Scan Mode	alphanumeric	1 byte	input
Tokenization Logic	alphanumeric	variable	input
Area Size	binary	4 bytes	output
Line Error	binary	4 bytes	output

Tokenization Logic

A set of statements which define the tokenization logic. All statements are maximal 80 characters long.

Syntax

$$\left\{\begin{array}{l}\textbf{IF = }\ \textit{class}\\ \textbf{IFMODE = }\ \textit{mode}\end{array}\right\}\left\{\left[\textbf{GOTO= }\ \textit{label}\right]\left[\textbf{ACTION= }\ \textit{action}\right]\left[\textbf{TYPE= }\textit{type}\right]\left[\textbf{CHAR= }\textit{c1}\right]\right\}$$

All parameters are keyword parameters (except for the label) and thus can be specified in any order.

Not all parameters must be declared, but each parameter can only be declared once per line; if multiple declarations of a parameter appear in the same line, then only the first parameter declaration is used.

The set is ended by a line containing the constant “END”.

IF=*class*

The value of *class* denotes the class of characters for which the statement is to be executed. This is the same class as that defined in the “classes” parameter of the SCTS call. If no IF is specified, the statement is executed once regardless of the class of the character.

IFMODE=*mode*

The value *mode* in the IFMODE parameter specifies whether an action is to be taken during text inversion or during query-syntax analysis. This parameter is applicable only if the scan mode is set to “A” (text inversion process and query syntax analysis). For other scan modes, this parameter is ignored. Possible values are:

T	Text inversion process only.
Q	Query-syntax analysis only.

If the parameter IFMODE is omitted, the action is taken during both text inversion and query syntax analysis.

GOTO=*label*

The value *label* in the GOTO parameter denotes the location (label) in a program from which execution is to be continued after the current statement is executed. If the parameter GOTO is omitted, execution continues with the next statement.

ACTION=*action*

The value *action* in the ACTION parameter denotes the action to be taken for the current character. Possible values for the parameter ACTION include:

Value	**Description**
ACCEPT	Character is to be accepted.
SKIP	Character is to be ignored.
BACK	Go back one character.
TRUNCATE	Current character string is to be truncated, thus forming a word.
REPLACE	Character is to be replaced by the character contained in the value “c1” of the parameter CHAR (see page 74).
INSERT	The character contained in the value “c1” of the parameter CHAR is to be inserted in the current character string *after* the current character.
MARK	Mark current character. This parameter is used in conjunction with parameter value BTOM (below).
BTOM	(Back To Mark) Go back to the character marked by ACTION=MARK. If no character is marked, go back to the start of token (i.e. erase the token).
INVERSE	TRUNCATE and invert the token.

TYPE=*type*

The value *type* of the TYPE parameter defines the type of token. The following token types can be specified.

Value	Description
NUMER	Numeric token.
STRING	Token is a string.

CHAR=*c1*

The value *c1* of the CHAR parameter denotes a one-byte character used together with the actions REPLACE or INSERT. If the value of the parameter ACTION is “REPLACE” or “INSERT”, a value for CHAR must be supplied. In all other cases CHAR is ignored.

Line Error

If an error is detected when the syntax of the character table is checked, this parameter contains the number of the line containing the error.

SCTX Call: Example 1

```
CALL 'TRS' 'SCTX' RETURN–CODE 'T'SCAN–LOGIC P–AREA–SIZE  LINE–ERROR
      WHERE SCAN–LOGIC IS A POINTER TO:

START     IF=SPEC,GOTO=SPEC1
          IF=APOS,GOTO=APOS1
          IF=ALPH,GOTO=SA,ACTION=ACCEPT
          IF=NUMBER,GOTO=SN,ACTION=ACCEPT
          IF=DOT,GOTO=DOT1
          ACTION=SKIP,GOTO=START
SA        IF=SEP,GOTO=SA,ACTION=SKIP
          IF=ALPH,GOTO=SA,ACTION=ACCEPT
          IF=NUMBER,GOTO=SA,ACTION=ACCEPT
          IF=DOT,GOTO=SA1,ACTION=ACCEPT
          ACTION=TRUNCATE
SA1       IF=ALPH,GOTO=SA,ACTION=ACCEPT
          IF=NUMBER,GOTO=SA,ACTION=ACCEPT
          ACTION=BACK
          ACTION=TRUNCATE
SN        IF=NUMBER,GOTO=SN,ACTION=ACCEPT
          IF=SEP,GOTO=SA,ACTION=SKIP
          IF=ALPH,GOTO=SA,ACTION=ACCEPT
          IF=DOT,GOTO=SN1,ACTION=ACCEPT
          ACTION=TRUNCATE,TYPE=NUMER
SN1       IF=ALPH,GOTO=SA,ACTION=ACCEPT
          IF=NUMBER,GOTO=SN,ACTION=ACCEPT
          ACTION=BACK
          ACTION=TRUNCATE,TYPE=NUMER
DOT1      IFMODE=Q,GOTO=DOT2
          ACTION=ACCEPT
          ACTION=TRUNCATE
DOT2      ACTION=SKIP,GOTO=START
SPEC1     IFMODE=Q,GOTO=SPEC2
          ACTION=SKIP,GOTO=START
SPEC2     IF=REL,GOTO=SPEC3
          ACTION=ACCEPT
          ACTION=TRUNCATE
```

```
SPEC3    ACTION=ACCEPT
         IF=EQ,ACTION=ACCEPT
         IF=REL,ACTION=ACCEPT
         ACTION=TRUNCATE

APOS1    IFMODE=Q,GOTO=APOS1A
         ACTION=SKIP,GOTO=START

APOS1A   ACTION=SKIP

APOS1B   IF=APOS,ACTION=SKIP,GOTO=APOS2
         ACTION=ACCEPT,GOTO=APOS1B

APOS2    ACTION=TRUNCATE,TYPE=STRING
         END
```

SCTX Call: Example 2

This example illustrates the translation of the German umlaut (Ü) to the anglicized version (UE).

```
IF=UMLAUT, ACTION=REPLACE,CHAR=U
ACTION=INSERT,CHAR=E,GOTO . . .
```

SCA – The Scan Routine

Description

The SCA routine tokenizes data by applying user-defined logic. It scans a free text line and returns one token at a time.

Note:
The SCA routine works only in the Adabas Text Retrieval application. A 'BC' call must precede this call.

Call Format

```
Call   'TRS'   'SCA'   parameters
```

Required Parameters	Format	Length	In/Output
Return Code	binary	4 bytes	output
Free Text Line	alphanumeric	variable	input
Line Length	binary	4 bytes	input
Disp in Text	binary	4 bytes	input, output
Token	alphanumeric	64 bytes	output
Token Type	binary	4 bytes	output
Token Length	binary	4 bytes	output
Scan Mode	alphanumeric	1 byte	input

Free Text Line

The text to be scanned.

Line Length

The length of the free text line in bytes.

Disp in Text

This is an internal variable within Adabas Text Retrieval. The current displacement in the text line must be zero with the first call to the scanner and must not be changed during successive calls within the same text line.

Token

Current token as found by the scanner.

Token Type

Type of token as defined in the scanner. Adabas Text Retrieval returns in this parameter the code of the type. The types used by the Adabas Text Retrieval system are:

Code	Detail
2	Numeric token
7	Token is a string

Token Length

Length of current token in bytes.

INCORE VOCABULARY

The incore vocabulary should contain high frequency words in order to reduce the number of Adabas calls and to speed up the document inversion process. To make such a incore vocabulary available to TRS, you must

- ensure that there is an existing vocabulary file with words in the V1 field;
- mark with an identifier all the words to be used by placing it in the V5 field;
- use the INCVOC parameter during a BC or DYP call to define the incore vocabulary. (To do this, you must use the same identifier which was used to mark the words in the vocabulary file.)

Warning:
*Do **not** use an incore vocabulary with the EXCLUDE option or together with the AUTOASP option. If you do, TRS ignores the INCVOC option if the EXCLUDE or AUTOASP option is used at the same time.*

Prerequisites for Using an Incore Vocabulary

- A vocabulary file where the V1 fields are filled
- The field V5 in the vocabulary file
- Determining which words belong to the incore vocabulary (up to a maximum of 1000)
- Determining the name of the vocabulary
- Putting the name of the vocabulary in the field V5 corresponding to the chosen words
- Defining the parameter ’INCVOC’ in the ’BC’ call

Example

ISN	V1	V5
1	THIS	MYWORDS
2	IS	MYWORDS
3	A	MYWORDS
4	COMPUTER	
5	ANOTHER	MYWORDS

Note: In order to put the name of the vocabulary in the field V5, you have to write your own program and let it run independently of Adabas Text Retrieval.

Document Indexing with Incore Vocabulary

With the prerequisites listed above, Adabas Text Retrieval can load the incore vocabulary. During the indexing process, Adabas Text Retrieval will first look in the incore vocabulary and if the word exists there, the system will get the word ISN from there; otherwise, it will look in the vocabulary file. This procedure reduces the Adabas calls and so the document indexing is faster.

USER-WRITTEN ROUTINES

Introduction to User-Written Routines

The User Functions, User exits, Phonetic Routine and Root Function are user written routines that are used in the TRS system. They are dynamically loaded at the startup of the TRS system during the BC call. Those functions must be present as dynamic shared library modules.

TRS allows up to three dynamic shared library modules. The user written routines must all be linked to up to three shared library modules. There is more than one shared library module because some of the routines are delivered with the TRS system, and sometimes the user wants to override a routine for a specific project.

To make the routines available to the TRS system do the following:

- Define the shared libraries as “trsuex1”, “trsuex2” or “trsuex3”.
- Build the shared libraries using the options of the C-compiler and linker on the corresponding platform.
- Copy the shared libraries to the directory “$TRSDIR/$TRSVERS”.

TRS looks for a routine first in the shared library “trsuex1”, then “trsuex2” and then in “trsuex3”. If a user wants to override a specific routine, he/she should put the new routine in the shared module that precede the shared module that the original routine is part of.

The entry point names of the user written routines must be:

Entry Point Name	**Target User Routine**
TRSUE01–TRSUE32	For user exits routine 1 – 32
TRSFFM0–TRSFFM9	For user function routines 0 – 9
TRSUFRT	For user root function routine.
TRSUPHO	For user phonetic function routine.

USER EXIT IN TRS QUERY

User Exits allow the user to intercept during execution of a query and to call TRS query functions from it.

1. User Exit as Part of the TRS System

- User Exit calling TRS (recursively) is part of the TRS current session. No new session (i.e., no new level) is defined, with the result that the recursive is done on a query basis.
- The User Exit is limited in the TRS calls it can perform. A TRS call like “BC” or “CL” is not recommended since it will change the TRS environment of the current TRS session. Since it is planned to be a fully recursive mechanism **on query level only**, TRS does not check for these kind of mistakes.
- All TRS calls used in the User Exits affect the whole system.
- TRS keeps the original values of some of the session parameters. The following parameters are saved before calling the User Exits and restored after returning from the User Exit: DFNR, DSFNR, VFNR, WORDLEN, SEARCHLB, THESAURI, DOCSET and WORDSET. Other parameters, if changed by the “DYP” call, will affect the session, even after returning from the User Exits.
- A “DSL” call to the same DFNR and DSFNR will override the previous definition.
- Adding documents inside the User Exit is not recommended since the User Exit does not know the status of the user application on a higher level.
- It is not recommended to use the TRS parameter ERRPRE. “ERRPRE” is a parameter given at the start of the session. It is a prefix to be added to TRS return code. The default setting is 0. Calling TRS from the User Exit will cause ERRPRE to be accumulated.

2. The TRS Call “DUE” – Define User Exit

To define the User Exit to TRS, we introduce a new TRS CALL named “DUE”. This call defines the User Exit search labels with the User Exits code number and the User Exit parameters passed from the application program to the User Exit via TRS. The ‘DUE’ call affects all the sessions. There is only one ‘DUE’ definition active in the system. A new ‘DUE’ will override the definition of the previous one.

The format of the call is:

CALL ‘TRS’ ‘DUE’ PARAMETERS

Required Parameters	**Format**	**Length**	**In/Output**
Return Code	binary	4 Bytes	output
Separation Value	alphanumeric	1 Byte	input
User Exit Definition	alphanumeric	variable	input

Return Code

The code returned at the end of processing, which indicates whether an error has occurred. A zero code indicates the normal end of processing.

Separation Value

A special character that defines the boundaries of the parameters passed to the User Exit. If this parameter is set to blank, a comma indicates end of parameters.

User Exit Definition

A free format text defining the User Exits search label, User Exit code number and the parameters passed to the User Exit from the application program via the TRS system.

The definition of the USER EXITS consist of a character string containing one or more entries separated by commas and ended by a period.

To define the USER EXIT, the name of the User Exit must be entered, followed by an equal sign and the User Exit parameters.

Example:

“UEX1 = parameters, DATE = parameters.”

Here we define two user exits: UEX1 and DATE.

The User Exits parameter consists of the User Exit code number, one or two bytes defining the User Exit code number (1–32), followed by a comma and the parameters pass to the User Exit via TRS starting with the Separation Value and ending with it.

Example:

CALL ’TRS’ ’DUE’ ’#’ ”HISTORY=5,#DSFNR=(4,5),GREGORIAN#.”

This is the definition of a User Exit called “HISTORY” (its search label), its User Exit number is 5 and the parameters passed to it are:

”DSFNR=(4,5),GREGORIAN”.

3. The Query Syntax:

The exit is identified by the search label defined in the “DUE” call. TRS isolates the sub-query for the exit and passes it to the user exit as is. No syntax checking or any other checking is done on that sub-query.

The sub-query boundaries are:

- from the exit search label to the next search label;
- from the exit search label to the next operator found in the query;
- from the exit search label to the end of the query;

Parentheses will also define the sub-query boundaries.

Examples:

1. **”HISTORY CATEGORY ASIA OR BOOKS CATEGORY ASIA”**

   Where HISTORY is a User Exit search label and BOOKS is a free text search label. TRS will call User Exit 5 and the User Exit will get the sub query “CATEGORY ASIA”.

2. **HISTORY (CATEGORY ASIA AND AFRICA) or HISTORY FRANCE.**

   TRS will call User Exit 5 twice: first, with sub query “CATEGORY ASIA AND AFRICA”; second, with sub query “FRANCE”.

4. The User Exit

The User Exit is able to call TRS and Adabas via the TRS mechanism. The exit can perform its own operation and then return to TRS with a request to finish executing the sub-query as part of the query itself. The exit dynamic area is allocated from the TRS common area via the TRS mechanism (The 'DSA' call). The exit may request TRS to release the dynamic area it allocated and to release a compiled query. The exit may not release a compiled query whose results are returned to TRS to be part of the query on a higher level.

5. Calling the User Exit

TRS calls the exits during query execution. For syntax checking only query ("ANAL" call), TRS calls the exits at the end of the syntax checking.

The exit input is:

- User Exit name (search label) as defined in the "DUE" call.
- The User Exit parameters passed to it from the application program, as written in the "DUE" call.
- The end user sub-query as written in the query itself.

The exit output is:

The User Exit output depends on the type of action TRS has to do with it. There are four types of output:

Abbreviation	File Type of Output
D	Result of Document Retrieval.
V	Result of Vocabulary Retrieval.
F	Values for Formatted Field Retrieval.
W	Values for Free Text Retrieval.

D: Result of Document Retrieval

The exit output:	Adabas Command id of an Adabas ISN-set from the document file, created by the User Exit and the number of ISN's contained in this ISN list.
TRS action:	Build a new ISN list identical to the User Exit ISN list for later reference.

V: Result of Vocabulary Retrieval:

The exit output:	Adabas Command Id of an Adabas ISN-set, from the vocabulary file, created by the User Exit and the number of ISN's contained in the ISN list.
TRS action:	Build a new ISN list identical to the exit ISN list for later reference. If it is a document retrieval query ('D' type), do the document retrieval.

F: Values for Formatted Field Retrieval:

The exit output:	One or more values to be retrieved. The formatted field Adabas name and the relational operator.
TRS action:	Do a formatted field retrieval according to exit parameters.

V: Values for Free Text Retrieval:

The exit output:	One or more values to be retrieved, the chapter Adabas name and the selection mode (type of selection on the vocabulary file).
TRS action:	Do a free text retrieval according to exit parameters.

FORMAT OF CALL

TRSUEnn (parameters). ‘nn’– is the User Exit code number. ‘nn’ is between 01 and 32. All calls are called by reference.

Required Parameters	Format	Length	In/Output
Pointer to TRS Common Area	Binary	4 Bytes	input
Return Code	Binary	4 Bytes	output
Action Code	Binary	1 Byte	input
User Exit Control Block	Binary	4 Bytes	input / output

TRS Common Area

TRS common area (known as TSIZE) needed in a direct call to TRS.

Note:
*TRS passes to the User exit a pointer to the common area. When calling TRS, a user should pass to the called routine the common area (in the “C” language it is *Pointer).*

Return Code

The code returned at the end of processing, by the exit, which indicates whether an error has occurred. A zero code indicates the normal end of processing. TRS adds the TRS error prefix to the return code and returns it to the application.

Action Code

One byte that is type of action:

A:	Only syntax checking is done.

X:	Syntax checking and execution is done.

User Exit Control Block

Address of a Control Block used to transfer information to and from the User Exit.

Field	Format	Length	In/Output	Output Type V D F W
Query Addr	binary	4 bytes	input	
Parm Addr	binary	4 bytes	input	
Query Length	binary	2 bytes	input	
Parm Length	binary	2 bytes	input	
Exit Name	alpha	32 bytes	input	
Exit Code	binary	1 byte	input	
VFNR DB No.	binary	2 byte	input	
VFNR File No.	binary	2 byte	input	
DFNR DB No.	binary	2 byte	input	
DFNR File No.	binary	2 byte	input	
DSFNR DB No.	binary	2 byte	input	
DSFNR File No.	binary	2 byte	input	
Query Type	alpha	1 byte	input	
Main Query Name	alpha	8 bytes	input	
Reserved		8 bytes		
Disp Error	binary	4 bytes	output	
Length Error	binary	4 bytes	output	
DFNR CID	binary	4 bytes	output	– R – –
DFNR QTY	binary	4 bytes	output	– R – –
VFNR CID	binary	4 bytes	output	R O O –
VFNR QTY	binary	4 bytes	output	R O O –
No. of Values	binary	2 bytes	output	– – R R

Field	Format	Length	In/Output	Output Type V D F W
Output Type	alpha	1 byte	output	R R R R
Query Name	alpha	8 bytes	output	O O O O
Field Name	alpha	2 bytes	output	R – R R
Value Format	alpha	1 byte	output	– – R R
Value Length	binary	1 byte	output	– – R R
DSA Name	alpha	8 bytes	output	O O R R
Function Code	alpha	1 byte	output	– – R R
Reserved		12 bytes		

R = required, O = optional, – = not used

Explanation of Fields

The fields in the preceding table are explained as follows:

Query Addr

Address of the end-user sub-query as written in the query itself.

Parm Addr

Address of application parameters to the User Exit, as written in the “DUE” call.

Query Length

The length of the sub-query.

Parm Length

Length of the application parameters to the User Exit.

Exit Name

User Exit search label as defined in the “DUE” call.

Exit Code

User Exit code number as defined in the “DUE” call.

VFNR DB No.

Adabas data base id of current vocabulary file.

VFNR File No.

Adabas file number of current vocabulary file.

DFNR DB No.

Adabas data base id of current documents file.

DFNR File No.

Adabas file number of current documents file.

DSFNR DB No.

Adabas data base id of current document index file.

DSFNR File No.

Adabas file number of current documents index file.

Query Type

The type of retrieval to be executed by the original QR call. There are two possible values:

D:	Documents retrieval
V:	Vocabulary retrieval.

Main Query Name

Name of TRS query that called the user exit.

Disp Error

If an error is detected when the syntax of the query is checked, the exit will return in this parameter the displacement of the erroneous entry within the query.

Length Error

If an error is detected when the syntax of the query is checked, the exit will return in this parameter the length of the erroneous entry within the query.

DFNR CID

The Adabas Command-ID of the Adabas ISN-set from document file as created by the User-Exit. TRS releases this Command-id at the end of the process.

DFNR QTY

The number of ISN's contained in the ISN-set identified by "DFNR CID".

VFNR CID

The Adabas Command-ID of the Adabas ISN-set from vocabulary file, as created by the User Exit. TRS releases this Command-id at the end of the process.

For "D" output type (DFNR CID) and "F" output type (FORMATTED FIELD VALUES), the "VFNR CID" is optional and is used by the highlight process during TRS CALL 'HIGH'.

VFNR QTY

The number of ISN's contained in the ISN-set identified by VFNR CID.

For "D" output type (DFNR CID) and "F" output type (FORMATTED FIELD VALUES) this parameter is optional and is used by highlight process during TRS CALL 'HIGH'.

Number of Values

Number of words for the TRS selection.

For 'F' and 'W' output type.

The User Exit puts the words in an area identified by "DSA-NAME".

Output Type

Abbreviation	Description
V	Result of vocabulary retrieval.
D	Result of document retrieval.
F	Values for formatted field retrieval.
W	Values for free text retrieval.

Query Name

Name of query to be released by TRS at the end of the process.

The exit can do TRS ‘QR’ calls. The exit may return the results of these calls (COMMAND-ID and quantity) to TRS for further processing.

The exit cannot do TRS ‘RQR’ call (release query) because TRS releases the result COMMAND-ID during ‘RQR’ call.

Field Name

Adabas name of a particular free text chapter or formatted field. TRS search is done on this field.

Value Format

Format of the values returns by the exit:

Abbreviation	Format of Values
A	Alphanumeric
B	Binary
P	Packed Decimal
U	Unpacked Decimal

Value Length

Length of every value in the area returns by the exit.

DSA-Name

The exit dynamic area is allocated from the TRS common area via the TRS ‘DSA’ call. “DSA-Name” identified the area in the TRS common area where the User Exit returns its values. At the end of the process TRS releases this area.

Function Code

Type of search to be made for free text chapters (for type ‘W’) or the relational operator for formatted field (for type ‘F’).

Function code for ‘W’ type:

Abbreviation	Function Code for W Type
W	STANDARD FUNCTION
S	SYNONYMS
X	SYNONYMS ROOT
R	ROOT
P	PHONETIC
C	CATEGORY
N	ASPECT – NARROW TERM
B	ASPECT – BROADER TERM
O	NEW SYNONYM MECHANISM
G	GROUP
0	USER FUNCTION 0
1	USER FUNCTION 1
2	USER FUNCTION 2
3	USER FUNCTION 3
4	USER FUNCTION 4
5	USER FUNCTION 5
6	USER FUNCTION 6
7	USER FUNCTION 7
8	USER FUNCTION 8
9	USER FUNCTION 9

Function code for ‘F’ type:

Abbreviation	Function Code for F Type
E	RELATIONAL EQUAL TO
G	RELATIONAL GREATER THAN
H	RELATIONAL GREATER OR EQUAL
L	RELATIONAL LESS THAN
S	RELATIONAL LESS OR EQUAL
B	RELATIONAL BETWEEN

USER FUNCTION IN TRS

The TRS system allows a user to define its own basic function, in addition to the standard function already defined in the system. This document describes the way a user function can integrate with the TRS system.

The system keywords of TRS are define in the TRS call ‘SCTW’. Ten user functions are defined there with the name “FUNC0” thru “FUNC9”. The user can add its own synonym to the user function name by calling the TRS call ‘SCTW’ (see Add Reserved Words in TRS Command Reference Manual).

The user function modules are named TRSFFMn where ‘n’ is the user function name (e.g., TRSFFM1).

TRS calls the user function with the following parameters: All calls are called by reference.

Requested Parameters	Format	Length	In / Output
Return Code	binary	4 bytes	output
Error Prefix	binary	4 bytes	input
Word	alphanumeric	variable	input
Word length	binary	4 bytes	input
Phonetic ID	alphanumeric	1 byte	input
Selection Code	alphanumeric	2 bytes	output
Word Vector	alphanumeric	variable	output
Number of Values	binary	4 bytes	output

Return Code

The code returned by the user function module at the end of processing which indicates whether an error has occurred. A zero code indicates normal end of processing. A non-zero code terminates the execution of the query process and the code is returned to the application program.

Error Prefix

A constant value to be added to all User Function return codes.

Word

Original word from the query itself. Its length is according to the parameter: Word Length.

Word Length

Length of original word and every entry in the vector of result words.

The user function returns the number of words returned in the result vector of words.

Phonetic ID

One byte Phonetic ID is given in the "DSL" call for the current chapter. If no Phonetic ID was given, this byte is left blank.

Selection Code

The user function returns the name of **Adabas** field from the vocabulary file to do the selection from, or a select code in the first byte of the parameter:

Adabas Filed / Code	Description
'W'	If search on words (including truncation).
'R'	If search on root.
'P'	If phonetic search.

For select code, the second byte must be left blank.

Word Vector

List of words generated by the user function. Its length is according to the parameter: Word Length.

Number of Values

Number of Values returned in the vector.

An example of a user function module:

Name: **TRSFFM1**
Function: **User Function 1 that returns the original word and asks TRS to search in the root field.**

```
int TRSFFM1
(
int    *prm_rc,         /* out      return code */
int    *prm_errp,       /* in       error prefix */
char *prm_word,         /* in       original word */
int    *prm_wlen,       /* in       word length */
char *prm_phonid,       /* in       phonetic id */
char *prm_code,         /* out      selection code */
char *prm_vector,       /* out      extended list of words */
int    *prm_cnt)        /* out      number of output values */
{
/* move original word to result vector */
memcpy(prm_vector,prm_word,*prm_wlen);
/* selection code is root */
*prm_code = 'R';
/* one word in result vector */
*prm_cnt = 1;
return 0;
}
```

THESAURUS / SYNONYM SYSTEM

Fields of Implemenation of TRS Thesaurus

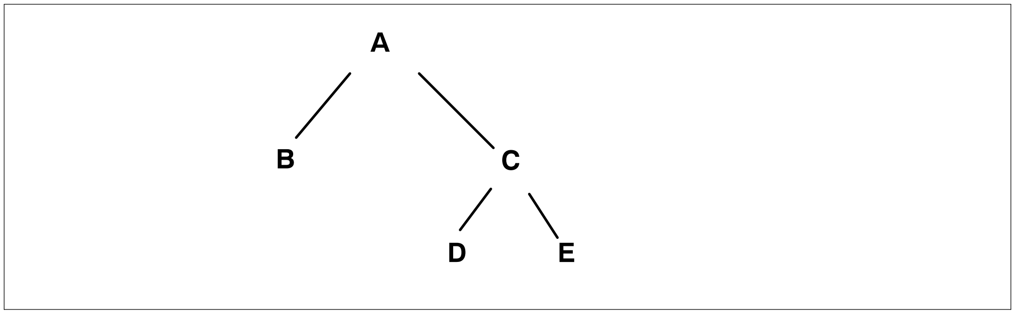
The implementation of TRS Thesaurus has the following fields:

Field	**Description**
1,T1,10,A,NU	THESAURUS ID – Identification of the thesaurus.
1,T2,64,A,NU	TREE ID – Identification of the thesaurus set.
1,T3,64,A,NU	TERM-BROAD – Broader term in case of a hierarchical relation or an entry of a synonym ring identified by the entry in the TREE-ID.
1,T4,64,A,NU	TERM-NARROW – Narrower term of a hierarchical relation or empty for a synonym term.
1,TC,80,A,NU	COMMENT – Comment for each term in the thesaurus
1,TD,10,A,NU	ORDER – Enable the user to view the thesaurus term in an ordered way.
T0 = T1(1,10),T2(1,64), T3(1,64)	Super-de – Used to access specific terms inside the thesaurus.
T9 = T1(1,10), T2(1,64), T4(1,64)	Super-de – Used to access specific terms inside the thesaurus
TW = T1(1,10), T2(1,64), T4(1,64), T3(1,64)	Super-de – Used to access specific terms inside the thesaurus
TW = T1(1,10), T2(1,64), T4(1,64), TD(1,10)	Super-de – Used to access specific terms inside the thesaurus + order
TX = T1(1,10), T3(1,64)	Super-de – Used to access specific terms inside the thesaurus
TY = T1(1,10), T4(1,64)	Super-de – Used to access specific terms inside the thesaurus

These fields define an aspect. This information is sufficient in order to define several thesaurus and to be able to isolate one “tree” from another by giving it TREE-ID. The user will call TRS to add this record. Adding a couple of records like that with the same THESAURUS-ID and the same TREE-ID defines a “tree”. In order to make the search more efficient we introduce another record type which serves us for building the HYPER DESCRIPTOR data:

Field	Description
1,T5,4,B,NU	TARGET ISN – ISN of the NARROW TERM as it is in the VFNR.
1,T6,18,A,NU	HYPER INFORMATION – Thesaurus id + ISN of the term in the VFNR file + level of tree.
1,T7,64,A,NU	TREE ID – Identification of the thesaurus set.
T8 = T6(3,12), T7(1,64)	Super De – Thesaurus Id + Tree Id
TA,18,A,NU = HYPER(n,T5,T6)	Hyper De – It’s value is the T6 field and it’s target ISN is the T5 value
TT = T6(3,12), T7(1,64), T5(1,4)	Super De – To allow GEN/UNGEN of a leaves.
TU = T6(3,12), T7(1,64), T6(13,16)	Super De – To allow GEN/UNGEN of a leaves.

The Hyper record is compiled out of the “tree” which was previously defined. There is a TRS call that compile a given set (“tree”) within a given thesaurus – e.g., like the following tree:



For vocabulary, we will have the words A, B, C, D, E. For example, the allocated ISN's are 1, 2, 3, 4, 5. The "tree" to be define will be as follows:

THESAURUS-ID	SET-ID	N-TERM	B-TERM
XX	YY	B	A
XX	YY	C	A
XX	YY	D	C
XX	YY	E	C

The compilation of it will generate the following records:

TARGET -ISN	REF - -TYPE	THESAU-RUS-ID	SET -ID	TERM	LEVEL
1	NT	XX	YY	A	00
2	NT	XX	YY	B	00
2	NT	XX	YY	A	01
3	NT	XX	YY	C	00
3	NT	XX	YY	A	01
4	NT	XX	YY	D	00
4	NT	XX	YY	A	02
4	NT	XX	YY	C	01
5	NT	XX	YY	E	00
5	NT	XX	YY	A	02
5	NT	XX	YY	C	01

The search on the HYPER will give us the required vocabulary ISN set (same as phonetic or ROOT). This structure demonstrates the "NT" (NARROW TERM) search, but in a similar way we implemented BROADER-TERM search (same "tree" definition but with additional HYPER records where, in REF-TYPE, we have "BT"). Also synonyms were implemented similarly.

The "level" field in the HYPER value allows us to search with a given "depth". In a regular search, we will search on range where level can be from "00" to "99" and, when a depth is requested, we will search as of the given level.

QR – Execute a Query

Search Modes Parameters

With the new Thesaurus/Synonym implementation there are three additional SEARCH MODE parameters:

- **NASPECTn**
- **TBTn**
- **NSYN**

NASPECT mode

Using the new Thesaurus implementation, all words are retrieved which are narrower terms occurring *n* levels lower in the Thesaurus hierarchy than the search term. Where *n* is omitted, all levels are retrieved.

TBTn

Using the new Thesaurus implementation, all words are retrieved which are broader terms occurring *n* levels higher in the Thesaurus hierarchy than the search term. Where *n* is omitted, all levels are retrieved.

NSYN

If search terms have synonyms defined for them in a Thesaurus, all documents containing the search term and its synonyms are found. The selection mode NSYN is based on the new synonym implementation.

The Thesaurus-id is defined in the DSL call to TRS. If no Thesaurus-id is defined in the DSL call for the current chapter, the default Thesaurus-id, as defined in the DYP/BC call, is used.

Thesaurus / Synonym Maintenance

To maintain the source records for Thesauruses and Synonyms there is a set of new **TRS** calls:

- **TADD** – Add a Connection/Synonym to a Set-ID.
- **TCHG** – Change a Father/Son Connection or Synonym.
- **TCOC** – Change Comment and/or Order of a Connection.
- **TDEL** – Delete a Father/Son Connection or Synonym.
- **TDID** – Deletion of a Set-ID in a Thesaurus.
- **TFAT** – List all Fathers of a Son in a Thesaurus.
- **TGEN** – Generate a Set-ID in a Thesaurus for Text Retrieval use.
- **TIDS** – List all Set-ID's in a Thesaurus.
- **TLST** – List all connections in a Set-ID.
- **TSET** – Check if a Set-ID exists on a Thesaurus.
- **TSON** – List all Sons of a Father in a Thesaurus.
- **TSYN** – List of all Sunonyms in a Set-ID.
- **TUNG** – Delete the Generation of a Set-ID in a Thesaurus.
- **TWRD** – Check if a term exists on a Set-ID.

TADD – Add a Connection/Synonym to a Set-ID

Description

The **TADD** call is used to add a father/son connection or a synonym to a Set-ID in a Thesaurus.

Format of Call

CALL 'TRS' 'TADD' Parameters

Requested Parameters	Format	Length	In / Output
Return Code	Binary	4 bytes	Output
Thesaurus Name	Alphanumeric	10 bytes	Input
Set-ID Name	Alphanumeric	64 bytes	Input
Father	Alphanumeric	64 bytes	Input
Son	Alphanumeric	64 bytes	Input
Comment	Alphanumeric	80 bytes	Input
Order	Alphanumeric	10 bytes	Input
Direction	Alphanumeric	1 byte	Input
Gen Done	Alphanumeric	1 byte	Output

Return Code

The code returned at the end of processing which indicates whether an error has occurred. A zero code indicates the normal end of processing.

RC =	251	–	The specified Son/Father connection already exists in this Set-ID.
RC =	252	–	The specified combination would create a loop.
RC =	258	–	Set-ID and addition must be the same type.
RC =	260	–	Son cannot be a blank field.

Thesaurus Name

Name of the Thesaurus in which the Set-Id is found (or will be found once the connection is added).

Set-ID Name

Name of the Set-Id in which the added connection will be found.

Father

The broader term in the added connection. (Empty in case of a synonym.)

Son

The narrower term in the added connection.

Comment

Each connection can have a user-comment added on to it.

Order

The order specified can be used when listing the sons of the father.

Direction

If this SET-ID is to be scanned upwards (besides downwards), this field should be set to “B” (Default = downwards only). In the case of SET-ID, which is a synonym type, this field is ignored.

Gen-Done

If the “Son” was a “Leaf”, the Gen-Done parameter will have a “Y”, meaning generation is taken care of; otherwise it will have a “N”, meaning that the user must deal with the regeneration.

TCHG – Change a Father/Son Connection or Synonym

Description

The TCHG call is used to make changes to an existing connection. The following fields are changed:

- SET-ID
- FATHER
- SON
- COMMENT
- ORDER

Format of Call

CALL ‘TRS’ ‘TCHG’ PARAMETERS

Requested Parameters	**Format**	**Length**	**In / Output**
Return Code	Binary	4 bytes	Output
Thesaurus Name	Alphanumeric	10 bytes	Input
Set-ID Name	Alphanumeric	64 bytes	Input
Father	Alphanumeric	64 bytes	Input
Son	Alphanumeric	64 bytes	Input
New Set-ID	Alphanumeric	64 bytes	Input
New Father	Alphanumeric	64 bytes	Input
New Son	Alphanumeric	64 bytes	Input
New Comment	Alphanumeric	80 bytes	input
New Order	Alphanumeric	10 bytes	Input
Direction	Alphanumeric	1 byte	Input
Gen-Done	Alphanumeric	1 byte	Output

Return Code

The code returned at the end of processing which indicates whether an error has occurred. A zero code indicates the normal end of processing.

RC =	252	–	The change would create a loop.
RC =	253	–	This Father/Son combination does not exist.
RC =	258	–	Change must be according to Set-ID type.
RC =	260	–	Set-ID or Son was blank.

Thesaurus Name

Name of Thesaurus in which the connection to be changed is found.

Set-ID Name

Name of the Set-ID in which the connection to be changed is found.

Father

The broader term of the connection to be changed. (Empty for synonyms.)

Son

The narrower term of the connection to be changed.

New Set-ID

The new Set-ID in which the changed connection will be found. If no change is requested, this field must hold the original Set-ID.

New Father

The new broader term of the changed connection. If no change is requested, this field must contain the original Father. (Empty for synonyms.)

New Son

The new narrower term of the changed connection. If no change is requested, this field must contain the original Son.

New Comment

The new comment for the changed connection. If no change is requested, this field must contain the original Comment.

New Order

The new order for the changed connection. If no change is requested, this field must contain the original order.

Direction

If the SET-ID is to be scanned upwards (in addition to downwards), this field should be set to “B” (Default = downwards only).

Gen-Done

If the “Son” was a “Leaf”, the Gen-Done parameter will have a “Y”; this means generation is taken care of. Otherwise it would have a “N”; this means that the user must deal with the regeneration. This parameter is applicable only for trees and not for synonyms. For synonyms, update to the generation is **never** done.

TCOC – Change Comment and/or Order of a Connection

Description

The **TCOC** call is used to make changes to the comment and/or order of an existing connection. This call does not cause need for regeneration.

Format of Call

CALL ‘TRS’ ‘TCOC’ PARAMETERS

Requested Parameters	**Format**	**Length**	**In / Output**
Return Code	Binary	4 bytes	Output
Thesaurus Name	Alphanumeric	10 bytes	Input
Set-Id Name	Alphanumeric	64 bytes	Input
Father	Alphanumeric	64 bytes	Input
Son	Alphanumeric	64 bytes	Input
New Comment	Alphanumeric	80 bytes	Input
New Order	Alphanumeric	10 bytes	Input

Return Code

The code returned at the end of processing which indicates whether an error has occurred. A zero code indicates the normal end of processing.

RC = 253 – This Father/Son combination does not exist.

Thesaurus Name

Name of Thesaurus in which the connection to changed is found.

Set-ID Name

Name of the Set-ID in which the connection to be changed is found.

Father

The broader term of the connection to be changed.

Son

The narrower term of the connection to be changed.

New Comment

The new comment for the changed connection. If no change is requested, this field must contain the original Comment.

New Order

The new order for the changed connection. If no change is requested, this field must retain the original order.

TDEL – Delete a Father/Son Connection or Synonym

Description

The TDEL call deletes a Father/Son connection or synonym from a Set-ID in a Thesaurus.

Format of Call

CALL ‘TRS’ ‘TDEL’ PARAMETERS

Requested Parameters	Format	Length	In / Output
Return Code	Binary	4 bytes	Output
Thesaurus Name	Alphanumeric	10 bytes	Input
Set-ID Name	Alphanumeric	64 bytes	Input
Father	Alphanumeric	64 bytes	Input
Son	Alphanumeric	64 bytes	Input
Gen-done	Alphanumeric	1 byte	Output

Return Code

The code returned at the end of processing which indicates whether an error has occurred. A zero code indicates the normal end of processing.

RC = 253 – This Father/Son combination or synonym does not exist.

Thesaurus Name

Name of Thesaurus in which the connection to be deleted is found.

Set-ID

Name of Set-ID in which the connection to be deleted is found.

Father

The broader term in the connection to be deleted. Empty in case of a synonym.

Son

The narrower term in the connection to be deleted.

Gen-Done

If the “Son” was a “Leaf”, the Gen-Done parameter will have a “Y”; this means generation is taken care of. Otherwise it will have a “N”; this means the user must deal with the generation. This parameter is applicable only for trees and not for synonyms. For synonyms, an update for the generation is never done.

TDID – Delete a all - from a Thesaurus

Description

The **TDID** call deletes the a SET-ID from a Thesaurus.

Format of Call

CALL ‘TRS’ ‘TDID’ PARAMETERS

Requested Parameters	**Format**	**Length**	**In / Output**
Return Code	Binary	4 bytes	Output
Thesaurus Name	Alphanumeric	10 bytes	Input
Set-ID	Alphanumeric	64 bytes	Input

Return Code

The code returned at the end of processing which indicates whether an error has occurred. A zero code indicates the normal end of processing. RC = 255 – This SET-ID/Thesaurus combination does not exist.

Thesaurus Name

Name of the Thesaurus in which the generator of the SET-ID to be deleted is found.

SET-ID

Name of SET-ID whose generation is to be deleted.

TFAT – List all Fathers found for a Son in a Thesaurus

Description

The **TFAT** call lists all Fathers found for a Son on all Set-ID's in a Thesaurus. Each call returns the next Father found.

Format of Call

CALL 'TRS' 'TFAT' PARAMETERS

Requested Parameters	**Format**	**Length**	**In / Output**
Return Code	Binary	4 bytes	Output
Thesaurus Name	Alphanumeric	10 bytes	Input
Set-ID	Alphanumeric	64 bytes	Input/Output
Father	Alphanumeric	64 bytes	Output
Son	Alphanumeric	64 bytes	Input
Step	Alphanumeric	1 byte	Input
Comment	Alphanumeric	80 bytes	Output
Order	Alphanumeric	10 bytes	Output
Work Area	Binary	4 bytes	*

Return Code

The code returned at the end of processing which indicates whether an error has occurred. A zero code indicates the normal end of processing.

RC =	255	–	No Fathers found.
RC =	263	–	End of list.

Thesaurus Name

The Thesaurus which will be scanned for the given Son.

Set-ID

The Set-ID in which the Father is found. If this field is not blank when **starting** this call, only Fathers in the specified Set-ID will be used.

Father

The father found.

Son

The Son whose Fathers is to be be listed.

Step

Three types of “step” exist:

- “S” – for start listing
- “N” - for next “Father”
- “E” - for end listing

Since this function returns one “Father” at a time, the user must specify at which stage he’s at, at the time of the call. It is suggested to call the function with “E” when listing is no longer required for freeing space.

Comment

The comment for the Father returned.

Order

The order for the Father returned.

Work Area

Internal Area, not to be touched by user.

Note:
When function called repeatedly (step = “N” or “E”) this same work-area must be passed.

TGEN – Generate Set-ID in Thesaurus for Text Retrieval

Description

After all necessary changes have been made to a Set-ID, the user must generate this Set-Id. This function deletes the old generation and creates a new one.

Format of Call

CALL ‘TRS’ ‘TGEN’ PARAMETERS

Requested Parameters	**Format**	**Length**	**In / Output**
Return Code	Binary	4 bytes	Output
Thesaurus Name	Alphanumeric	10 bytes	Input
Set-ID Name	Alphanumeric	64 bytes	Input
Direction	Alphanumeric	1 byte	Input

Return Code

The code returned at the end of processing which indicates whether an error has occurred. A zero code indicates the normal end of processing.

RC = 255 – This Set-ID/Thesaurus combination does not exist.

Thesaurus Name

Name of the Thesaurus in which the Set-ID to be generated is found.

Set-ID

Name of the Set-ID to be generated.

Direction

If this SET-ID is to be scanned upwards (besides downwards), this field should be set to: “B”. (Default = downwards only.) In the case of a Set-ID which is a synonym type, this field is ignored.

TIDS – List All Set-ID's in a Thesaurus

Description

The **TIDS** call lists all Set-ID's found in a Thesaurus. this function is called repeatedly to get all Set-ID's. After each call, one Set-ID is returned.

Format of Call

CALL 'TRS' 'TIDS' PARAMETERS

Requested Parameters	Format	Length	In/Output
Return Code	Binary	4 bytes	Output
Thesaurus Name	Alphanumeric	10 bytes	Input
Step	Alphanumeric	1 byte	Input
Set-ID	Alphanumeric	64 bytes	Output
Set-ID Type	Alphanumeric	1 byte	Output
Work Area	Binary	68 bytes	*

Return Code

The code returned at the end of processing which indicates whether an error has occurred. A zero code indicates the normal end of processing.

RC =	255	–	No Set-ID's found.
RC =	263	–	End of list.
RC =	257	–	No start has been done.

Thesaurus Name

The Thesaurus which is to have its Set-ID's listed.

Step

Three types of “step” exist:

- “S” – for start listing
- “N” – for next Set-ID
- “E” – for end listing

Since this function returns one Set-ID at a time, the user must specify at which stage he’s at, at the time of the call. It is suggested to call the function with “E” when listing is no longer required for freeing space.

Set-ID

The Set-ID found.

Set-ID Type

The type of Set-ID. “S” for Synonym.

“T” for Tree.

Work Area

Internal work area – not to be touched by user!

Note:
When function called repeatedly (step = “N” or “E”) this same work-area must be passed.

TLST – List all Connections in a Set-ID

Description

The **TLST** call lists all connections found in a Set-ID. This function is called repeatedly, after each call the next connection is returned. The order of the connections listed is by depth, meaning: A "grandchild" of a son will appear before that son's "brother" appears.

Format of Call

CALL 'TRS' 'TLST' PARAMETERS

Requested Parameters	**Format**	**Length**	**In / Output**
Return Code	Binary	4 bytes	Output
Thesaurus Name	Alphanumeric	10 bytes	Input
Set-ID Name	Alphanumeric	64 bytes	Input
Top Word	Alphanumeric	64 bytes	Input
Step	Alphanumeric	1 byte	Input
Order Type	Alphanumeric	1 byte	Input
Father	Alphanumeric	64 bytes	Output
Son	Alphanumeric	64 bytes	Output
Comment	Alphanumeric	80 bytes	Output
Order	Alphanumeric	10 bytes	Output
Level	Binary	4 bytes	Output
Work-Area	Binary	72 bytes	*

Return Code

The code returned at the end of processing which indicates whether an error has occurred. A zero code indicates the normal end of processing.

RC =	263 –	End of list.
RC =	255 –	Set-ID not found.
RC =	261 –	Set-ID is a synonym type one.
RC =	257 –	List hasn't been started.

Thesaurus Name

Name of Thesaurus to be listed.

Set-ID Name

Name of the Set-ID to be listed.

Top Word

Optional. If this field is not blank the listing will start from this term, otherwise the listing starts from the top of the tree. This field should not be touched after the "start" step.

Step

Three types of "step" exist:

- "S" – for start listing
- "N" – for next connection
- "E" – for end listing

Since this function returns one connection at a time, the user must specify at which stage he's at, at the time of the call. It is suggested to call the function with "E" when listing is no longer required for freeing space.

Order Type

The user can get the listing by internal “order” (using the “order field”) or by alphabetical order. This field must get either: “I” – internal or “A” – alphabetical (default).

Father

The broader term of the connection returned.

Son

The narrower term of the connection returned.

Comment

The comment of the connection returned.

Order

The order of the connection returned.

Level

The relative level of the connection returned: relative to the “top word” if one specified otherwise, relative to the top of the tree.

Work area

Internal work area, not to be touched by the user!

Note:
When function called repeatedly (step = “N” or “E”) this same work-area must be passed.

TSET – Check if a Set-ID exists in a Thesaurus

Description

The **TSET** call checks if a Set-ID exists in a given Thesaurus.

Format of Call

CALL ‘TRS’ ‘TSET’ PARAMETERS

Requested Parameters	**Format**	**Length**	**In / Output**
Return Code	Binary	4 bytes	Output
Thesaurus Name	Alphanumeric	10 bytes	Input
Set-ID Name	Alphanumeric	64 bytes	Input
Set Type	Alphanumeric	1 byte	Output
Found	Alphanumeric	1 byte	Output

Return Code

The code returned at the end of processing which indicates whether an error has occurred. A zero code indicates the normal end of processing.

Thesaurus Name

Thesaurus in which the Set-ID is to be looked for.

Set-ID Name

Name of Set-ID to be looked for.

Set Type

If Set-IS found, this field will contain either: ‘T’ – for “tree” or ‘S’ – for “synonym”.

Found

Will contain either ‘Y’ (for “yes” found) or ‘N’ (for **not** found).

TSON – List all Sons of a Father in Thesaurus

Description

The **TSON** call lists all Sons found for a Father in all Set-ID’s in a Thesaurus. Each call returns the next Son found.

Format of Call

CALL ‘TRS’ ‘TSON’ PARAMETERS

Requested Parameters	**Format**	**Length**	**In / Output**
Return Code	Binary	4 bytes	Output
Thesaurus Name	Alphanumeric	10 bytes	Input
Set-ID	Alphanumeric	64 bytes	Input/Output
Father	Alphanumeric	64 bytes	Input
Son	Alphanumeric	64 bytes	Output
Step	Alphanumeric	1 byte	Input
Order Type	Alphanumeric	1 byte	Input
Comment	Alphanumeric	80 bytes	Output
Order	Alphanumeric	1 byte	Output
Work Area	Binary	4 bytes	*

Return Code

The code returned at the end of processing which indicates whether an error has occurred. A zero code indicates the normal end of processing.

RC =	255 –	No Sons found.
RC =	263 –	End of List.

Thesaurus Name

The Thesaurus Name which will be scanned for the given Father.

Set-ID

The Set-ID in which the Son is found. If this field is not blank when **starting** this call, only sons in the specified Set-ID will be listed.

Father

The father whose sons are to be listed.

Son

The Son found.

Step

Three types of "step" exist:

- "S" – for start listing
- "N" – for next "Son"
- "E" – for end listing

Since this function returns one "Son" at a time, the user must specify at which stage he's at, at the time of the call. It is suggested to call the function with "E" when listing is no longer required for freeing space.

Order Type

The user can get the listing by internal "order" (using the "order field") or by alphabetical order. This field must get either: "I" (internal) or "A" (alphabetical), by default.

Comment

The comment for the Son returned.

Order

The order for the Son returned.

Work Area

Internal Area, not to be touched by user.

Note:
When function called repeatedly (step = "N" or "E") this same work-area must be passed.

TSYN – List all Synonyms in a Set-ID

Description

The **TSYN** call lists all Synonymous found in a Set-ID. This function is called repeatedly, after each call the next Synonym is returned.

Format of Call

CALL 'TRS' 'TSYN' PARAMETERS

Requested Parameters	Format	Length	In / Output
Return Code	Binary	4 bytes	Output
Thesaurus Name	Alphanumeric	10 bytes	Input
Set-ID Name	Alphanumeric	64 bytes	Input
Step	Alphanumeric	1 byte	Input
Synonym	Alphanumeric	64 bytes	Output
Comment	Alphanumeric	80 bytes	Output
Work Area	Binary	4 bytes	*

Return Code

The code returned at the end of processing which indicates whether an error has occurred. A zero code indicates the normal end of processing.

RC = 263 –	End of List.
RC = 255 –	Set-ID not found
RC = 257 –	List hasn't been started.
RC = 259 –	Set-ID not of a Synonym type

Thesaurus Name

Name of Thesaurus to be listed.

Set-ID Name

Name of Set-ID to be listed.

Step

Three types of “step” exist:

- “S” – for start listing
- “N” – for next connection
- “E” – for end listing

Since this function returns one connection at a time, the user must specify at which stage he’s at, at the time of the call. It is suggested to call the function with “E” when listing is no longer required for freeing space.

Synonym

The Synonym returned.

Comment

The comment of the connection returned.

Work-Area

Internal work area – not to be touched by the user!

Note:
When function is called repeatedly (step = “N” or “E”), this same work must be passed.

TUNG – Delete the Generation of a Set-ID in a Thesaurus

Description

The **TUNG** call deletes the generation of a Set-ID.

Format of Call

CALL ‘TRS’ ‘TUNG’ PARAMETERS

Requested Parameters	**Format**	**Length**	**In / Output**
Return Code	Binary	4 bytes	Output
Thesaurus Name	Alphanumeric	10 bytes	Input
Set-ID Name	Alphanumeric	64 bytes	Input

Return Code

The code returned at the end of processing which indicates whether an error has occurred. A zero code indicates the normal end of processing.

Thesaurus Name

Name of the Thesaurus in which the generator of the Set-ID to be deleted is found.

Set-ID

Name of the Set-ID whose generation is to be deleted.

TWRD – Check if a Word exists in a Set-ID

Description

The **TWRD** call checks if a specified word exists in a Set-ID.

Format of Call

CALL ‘TRS’ ‘TWRD’ PARAMETERS

Requested Parameters	**Format**	**Length**	**In / Output**
Return Code	Binary	4 bytes	Output
Thesaurus Name	Alphanumeric	10 bytes	Input
Set-ID Name	Alphanumeric	64 bytes	Input
Word	Alphanumeric	64 bytes	Input
Found	Alphanumeric	1 byte	Output

Return Code

The code returned at the end of processing which indicates whether an error has occurred. A zero code indicates the normal end of processing.

Thesaurus Name

The Thesaurus in which the word is being checked for.

Set-ID Name

Name of the Set-ID in which the word is being checked for.

Word

The word to be checked for.

Found

Either a ‘Y’ (for ‘YES’– word was found) or a ‘N’ (for ‘NO’ – not found) is returned in this field.

FILE STRUCTURE

The information in Adabas Text Retrieval is stored in three logical Adabas files which can be stored in one or more physical Adabas files. The three logical Adabas files are:

- Document file (DFNR);
- Vocabulary file (VFNR);
- Document index file (DSFNR).

The document file must contain user-defined formatted fields which are to be used for retrieval operations. The Adabas ISNs of the document file will make up the resulting ISN sets for retrieval operations.

The vocabulary file contains the word index and thesaurus of Adabas Text Retrieval.

The document index file contains the indices of all free-text chapters.

Important:

Because the Adabas ISNs of all files mentioned above are used by Adabas Text Retrieval for free-text indexing, the Adabas ISNs must not be deleted or changed in any way. If any of these files is reloaded, the USERISN parameter must be specified.

An Example of the Index Structure

The following diagrams show how the internal index structure of Adabas Text Retrieval functions by way of a simple example, and how Adabas Text Retrieval indices are used when a retrieval operation is carried out.

For example, in Figure 9-1 the two documents DOC1 and DOC2 are allocated the ISNs 678 and 679 in the document file.

Document File

ISN	DOCID	TEXT
678	DOC1	this is a computer
679	DOC2	this is another computer

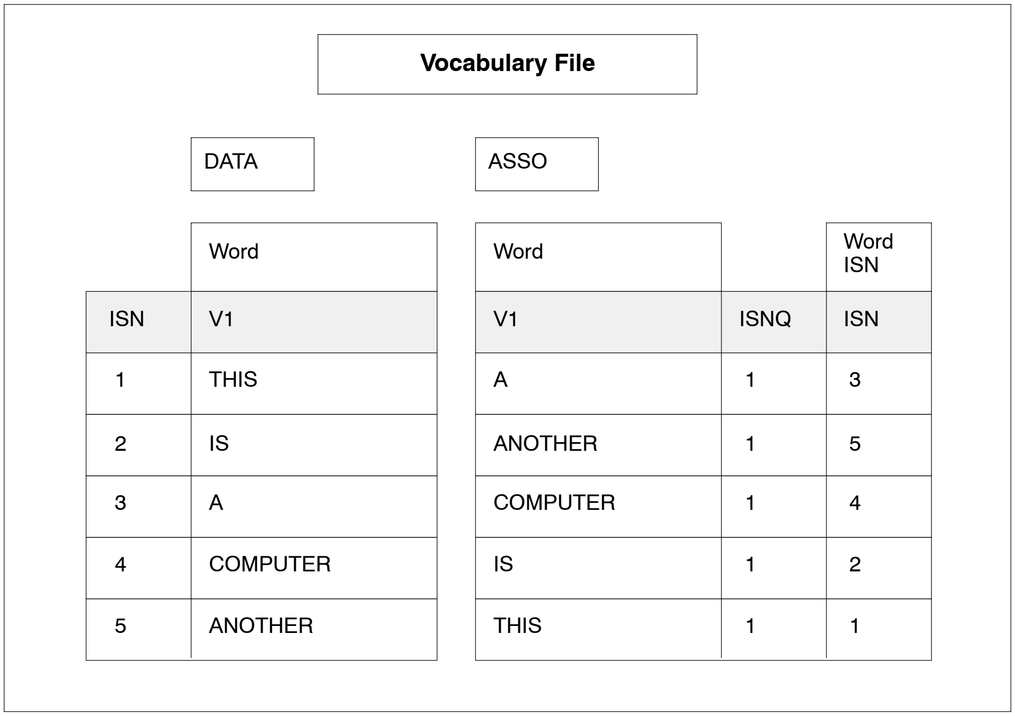
Figure 9-1

During the inversion process, each word of a document is entered on the vocabulary file (see Figure 9-2). The vocabulary file, similar to a dictionary, contains a *single* entry of each word known to the system no matter how frequently the word occurs in the documents. The words known to the system are entered in the V1 field. Each word entered receives an Adabas ISN unique to that word. This unique Adabas ISN is called a word ISN.

Thereafter, the name of the free-text chapter to which the information belongs is entered in the D1 field on the document index file (see Figure 9-3). The document ISN of the document to which the information belongs is entered in the D0 field. The word ISNs representing the words of which the document consists are entered in the multiple field D3.

The Adabas hyperdescriptor technique enables Adabas Text Retrieval to make up an inverted list which relates the word ISNs of a document to the document ISN of the document in question, although this document ISN is part of the data of the document index file and not the original ISN as assigned by Adabas.

During a retrieval operation (see Figure 9-4) Adabas Text Retrieval searches the vocabulary file for the ISN of the word sought. In the example on the following pages, the word “computer” is sought. If the word cannot be found, the result of the query will be zero; otherwise Adabas Text Retrieval uses the word ISN (in this case 4) to find all documents containing the word sought on the document index file. In this case, the documents with the document ISNs 678 and 679 are in the document file.



Vocabulary File

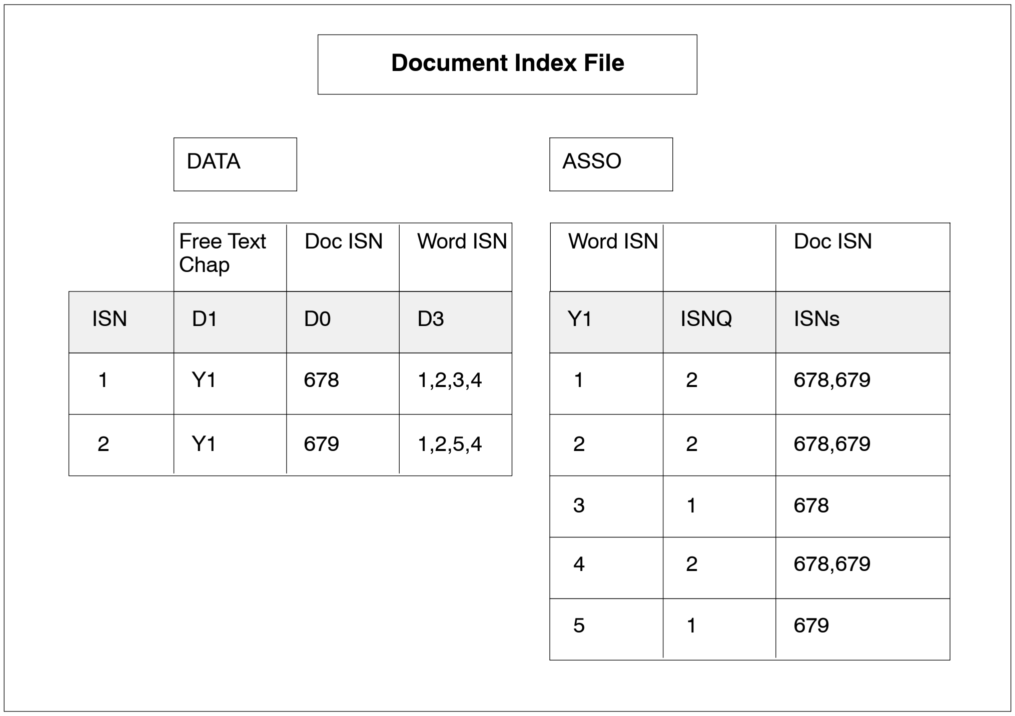
DATA

	Word
ISN	V1
1	THIS
2	IS
3	A
4	COMPUTER
5	ANOTHER

ASSO

Word		Word ISN
V1	ISNQ	ISN
A	1	3
ANOTHER	1	5
COMPUTER	1	4
IS	1	2
THIS	1	1

Figure 9-2



Document Index File

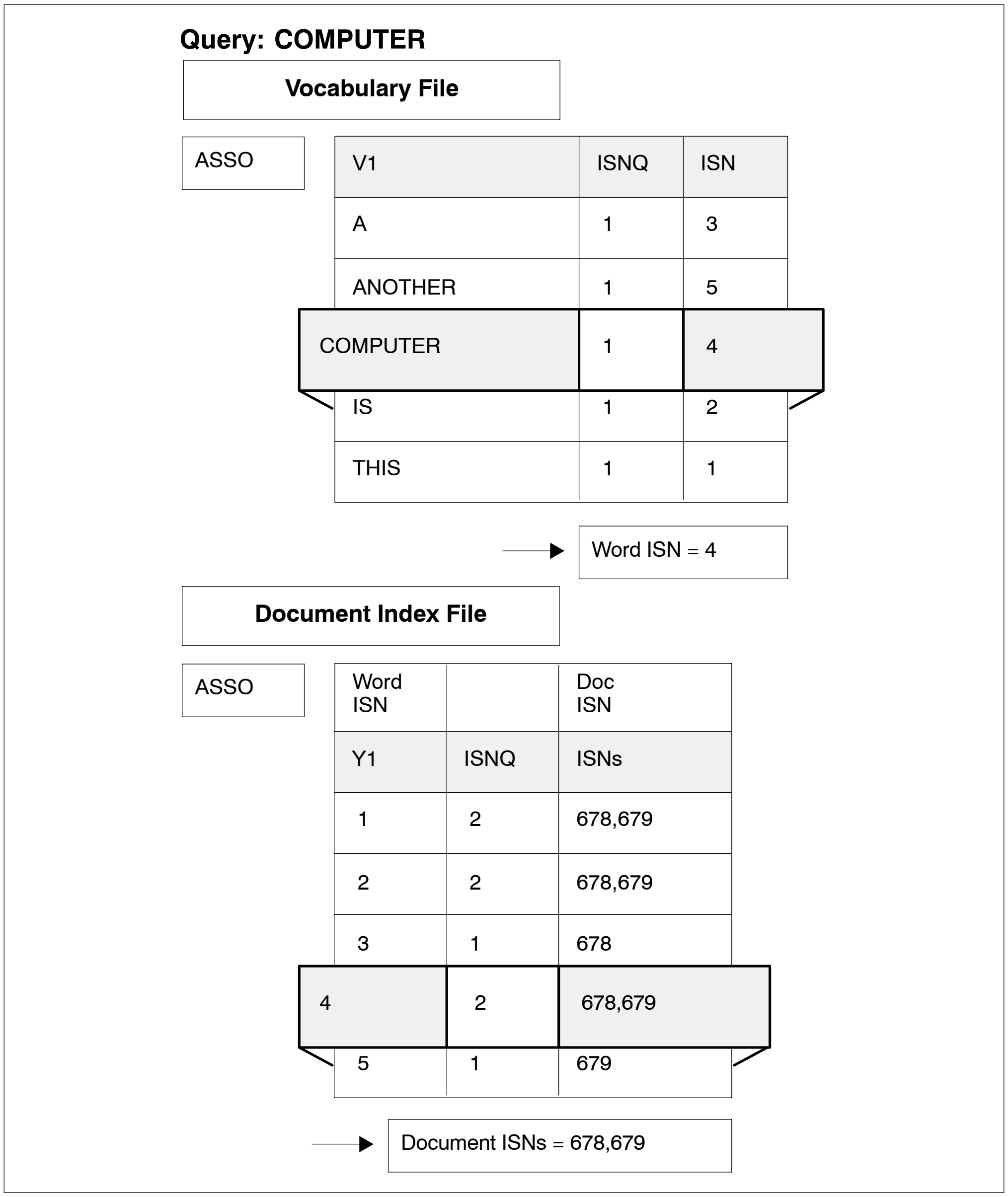
DATA

	Free Text Chap	Doc ISN	Word ISN
ISN	D1	D0	D3
1	Y1	678	1,2,3,4
2	Y1	679	1,2,5,4

ASSO

Word ISN		Doc ISN
Y1	ISNQ	ISNs
1	2	678,679
2	2	678,679
3	1	678
4	2	678,679
5	1	679

Figure 9-3



Query: COMPUTER

Vocabulary File

ASSO

V1	ISNQ	ISN
A	1	3
ANOTHER	1	5
COMPUTER	1	4
IS	1	2
THIS	1	1

→ Word ISN = 4

Document Index File

ASSO

Word ISN		Doc ISN
Y1	ISNQ	ISNs
1	2	678,679
2	2	678,679
3	1	678
4	2	678,679
5	1	679

→ Document ISNs = 678,679

Figure 9-4

Document File

Adabas Text Retrieval uses the Adabas ISNs of the document file to identify the documents. These Adabas ISNs form the result of any query entered in the system in the form of Adabas ISN sets. The respective Adabas ISN is either identified by the value of a unique formatted field on the document file or entered directly as a parameter to the ADD or DDS call (see the DOCID parameter in the DYP call, page 40).

In order to perform queries accessing the contents of the free-text chapters together with user defined formatted fields all relevant formatted fields must be contained in the document file.

For formatted fields you have to specify the corresponding descriptors. For example, for the field ORDER (A16,N,D) in the demo application:

’01,DA,16,A,DE,NU’

Vocabulary File

The vocabulary file contains:

- The vocabulary (word index) of Adabas Text Retrieval;
- The Adabas Text Retrieval thesaurus.

It consists of the following fields:

Field	Description
1,V1,32,A,DE,NU	(WORD) The field V1 contains the standardized word as encountered by Adabas Text Retrieval during the inversion process. A typical case of standardization is the translation of all characters (letters) contained in a word to upper case.
1,V2,32,A,DE,NU,MU	(WORD-DOUBLE) The field V2 contains the internal index value for the execution of double truncation (*string*).
1,V3,32,A,DE,NU	(WORD-REVERSE) The field V3 contains the internal index values for the execution of left and middle truncation.

Field	Description
1,V4,32,A,DE,MU,NU	(ROOT) The field V4 contains user defined roots of the word contained in the V1 field; it supports the execution of the ROOT or SYR selection modes.
1,V5,32,A,DE,MU,NU	(ASPECT) The field V5 contains user-defined, broader terms (ASPECTS) of the word contained in the V1 field: it supports the execution of the ASPECT and GROUPn selection modes.
V6=PHON(V1)	(PHONETIC) The phonetic descriptor V6 is built on the basis of the word contained in the V1 field. It supports the execution of the PHONETIC selection mode.
or:	
1,V6,32,A,DE,MU,NU	(PHONETIC) The field V6 contains the phonetic value built on the basis of the word contained in the V1 field with a specific user exit in Adabas Text Retrieval.
1,V8,32,A,DE,MU,NU	(SYNONYM) The occurrences of the multiple value field V8 make up a synonym ring containing those user defined words which are of equal meaning; they support the execution of SYN or SYR selection modes.
1,V9,32,A,NU	(ORIGINAL) The field V9 contains the original nonstandard form of the word as contained in the V1 field and as encountered by Adabas Text Retrieval during the inversion process at its first occurrence in any text entered into the system. If the original non-standardized form of the word is not required, the field V9 can be omitted.

Note:
The length of all the fields above corresponds to the word length inside Adabas Text Retrieval. It is specified by the WORDLEN parameter of the BC or DYP call. It must not exceed 64 bytes. The default word length is 32 bytes.

Document Index File

The document index file contains the internal document index created by Adabas Text Retrieval during the inversion process.

It consists of the following fields:

Field	Description
1,D0,4,B,NU	ISN of DFNR record
1,D1,2,A,NU	Hypername
1,D2,191,A,NU,MU	Proximity information
1,D3,4,B,NU,MU	Word ISN in VFNR
1,D7,12,B,NU	Paragraph, sentence and word position
D9=D0(1,4),D1(1,2),D7(5,6)	

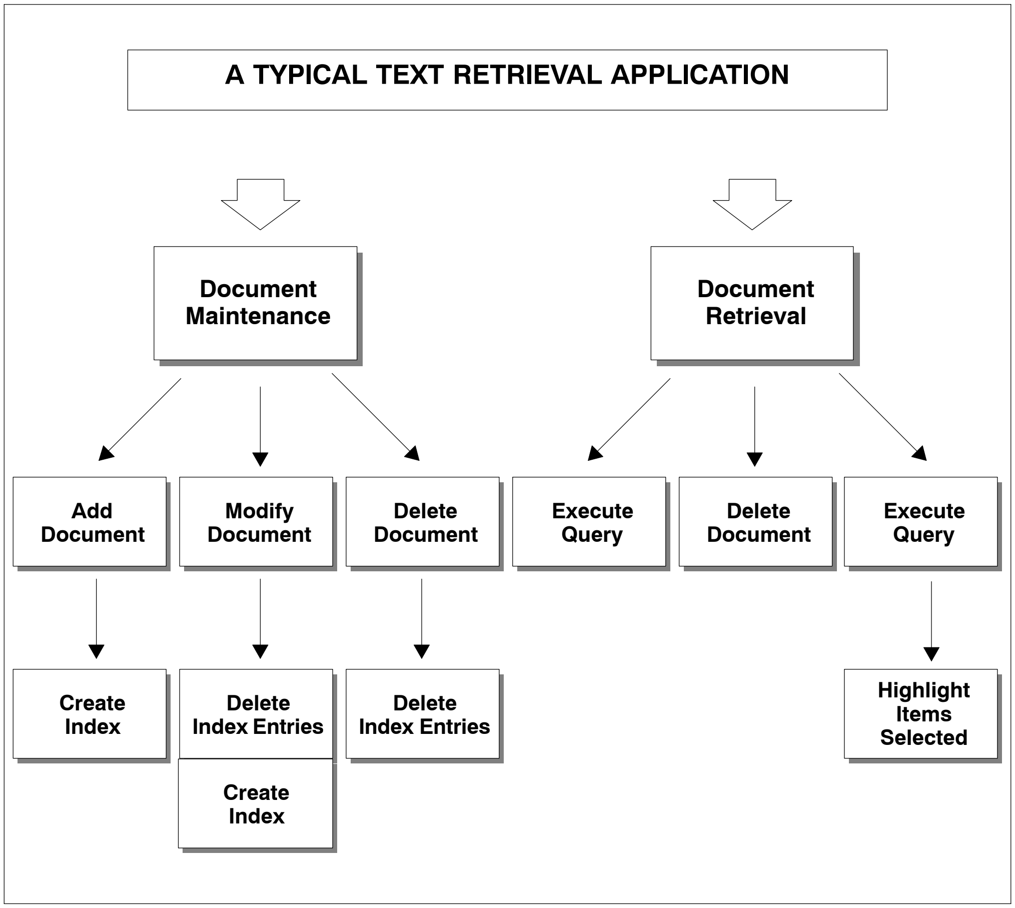
For each user-defined, free-text chapter, a hyperdescriptor in the following form must be added to the document index file:

Field	Description
Y1,4,B,NU,MU=HYPER(1,D1,D0,D3)	This hyperdescriptor definition should be used for applications which do not use proximity search.
Y1,8,B,NU,MU=HYPER (1,D1,D0,D7,D2,D3)	This hyperdescriptor definition should be used for applications which use proximity search.

SAMPLE APPLICATION

This chapter demonstrates the use of Adabas Text Retrieval calls in the context of a sample application. This sample application is written in Natural and is contained as a Natural INPL on the Adabas Text Retrieval installation tape.

A retrieval application typically consists of the functions illustrated below:

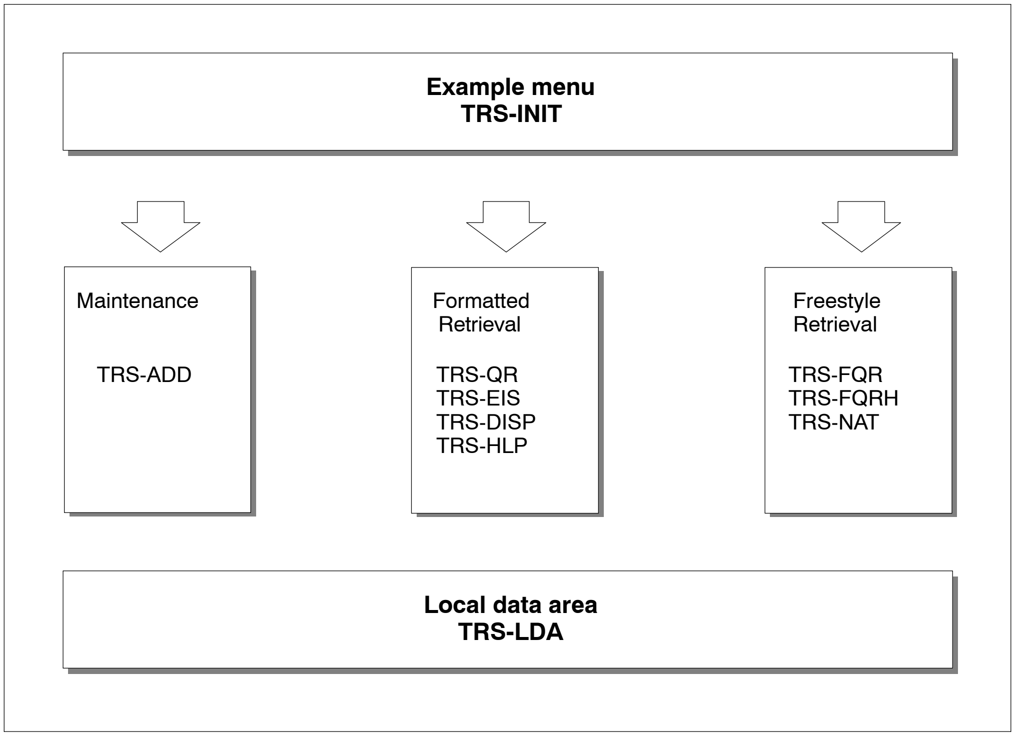


The sample application explained in this chapter is a small text retrieval system used to retrieve information on Software AG's product documentation. It enables the maintenance and retrieval of documents.

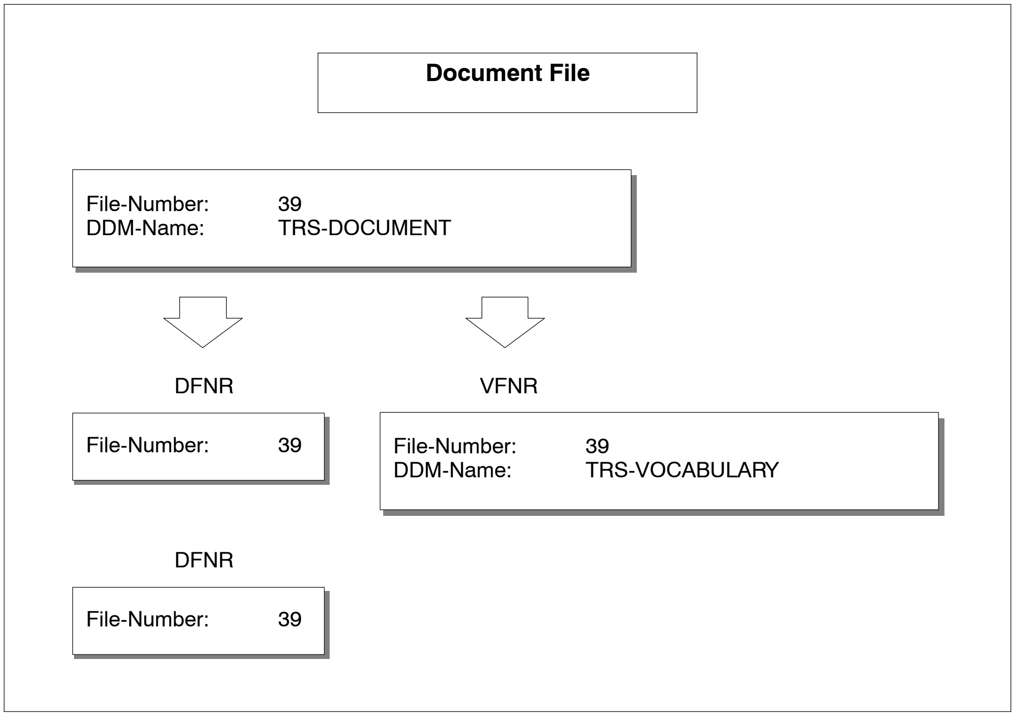
The following main functions are implemented:

- **Maintenance.** Store, update, delete documents;
- **Formatted retrieval.** Query, select, display documents;
- **Freestyle retrieval.** Query, display, overview.

The following figure provides an overview of the sample application:



The following diagram provides an overview of the sample application file structure:



To implement the sample application, you must adjust the physical file numbers to the values chosen by you when installing the sample application files in the Adabas database:

- Change the file numbers of the DDMs TRS-DOCUMENT and TRS-VOCABULARY;
- Change the file numbers used for the Adabas Text Retrieval BC call in the Natural program TRS-INIT (statement number 0330 , page 142);
- Recatalog all Natural objects with the Natural CATALL command.

The Calls Used in the Sample Application

The following table alphabetically lists the calls used in the sample application:

Call	**Description**	**In Sample Program**
ADD	Inverts a document	**TRS-ADD**
BC	Starts an Adabas Text Retrieval session	**TRS-INIT**
CL	Closes an Adabas Text Retrieval session	**TRS-INIT**
DDS	Deletes document index entries	**TRS-ADD**
DSL	Defines search labels	**TRS-INIT**
DYP	Changes dynamic parameters	**TRS-ADD** **TRS-DISP**
EISE	Ends browsing through an ISN set	**TRS-EIS**
EISG	Browses through an ISN set	**TRS-EIS**
EISS	Starts browsing through an ISN set	**TRS-EIS**
HIGH	Highlights a document	**TRS-DISP**
QR	Executes a query	**TRS-HLP** **TRS-FQR** **TRS-QR**
RQR	Releases a query	**TRS-FQR** **TRS-QR**

Initialize Adabas Text Retrieval Session

TRS-INIT

The program TRS-INIT initializes an Adabas Text Retrieval session and invokes the required function (maintenance, retrieval, freestyle retrieval). The program is broken down into the following functional segments.

- Initialize program;
- Set to lower case;
- Initialize the Adabas Text Retrieval Session (BC call);
- Define Search Labels (DSL call);
- Display Menu and Invoke Selected Functions;
- Close the Adabas Text Retrieval Session (CL call).

Program Start Up

```
***********************************************************************
*                                                                     *
*  ADABAS TEXT RETRIEVAL  Example Application                         *
*                                                                     *
*  Object   :  TRS-INIT                                               *
*  Type     :  Program                                                *
*  Function :  Initialize TRS session                                 *
*  Author   :  Software AG                                            *
*                                                                     *
***********************************************************************
*
DEFINE DATA LOCAL USING TRS-LDA
            LOCAL
1 SEL (A1)
END-DEFINE
*
```

Use Lowercase letters

```
* ------------------------------------------------------------------- *
*  Set lower case                                                     *
* ------------------------------------------------------------------- *
SET CONTROL 'L'
*
```

Initialize the Adabas Text Retrieval Session

Note:
It is recommended that you execute a CL call prior to a BC call to ensure that the existing session is closed.

The BC call in line 0330 is used to initialize an Adabas Text Retrieval session.

The TRS.DYP-PARM parameter contains all relevant Adabas Text Retrieval start-up parameters for that specific session.

```
0220 * -------------------------------------------------------------------- *
0230 *  Initialize TRS session                                               *
0240 * -------------------------------------------------------------------- *
0250 *
0260 *
0270 CALL 'TRS' 'CL' TRS.RC TRS.SAVE
0280 *
0290 *
0300 MOVE 'VFNR=38,DFNR=39,DSFNR=39,DOCID=DA,SETCHAR=#.' TO TRS.DYP-PARM
0310 *
0320 *
0330 CALL 'TRS' 'BC' TRS.RC TRS.SIZE TRS.SAVE TRS.DYP-PARM
0340 *
0350 *
0360 IF TRS.RC NE 0
0370    MOVE TRS.RC TO TRS.RC1
0380 *
0390 *
0400    CALL 'TRS' 'CL' TRS.RC TRS.SAVE
0410 *
0420 *
0430    WRITE  'Error in TRS-BC:' TRS.RC1
0440    STOP
0450 END-IF
0460 *
```

Define Search Labels

In order to use logical names for the retrieval of free-text chapters and formatted fields in the document structure, search labels are assigned to these fields using the DSL call (see line 0530). The parameter TRS.DSL-PARM must contain all necessary search label definitions.

The logical name #TITLE is assigned to the free-text chapter hyperdescriptor Y1Y1 and the logical name #ABSTRACT to Y2Y2 (see line 0500).

The logical name #DATE is assigned to the formatted field represented by the Adabas descriptor DB and the logical name #ORDER to DA (see line 0500).

These logical names can be used for processing queries which access the requested fields in the document structure.

```
0470 * ------------------------------------------------------------------- *
0480 *  Define Search-Labels                                                *
0490 * ------------------------------------------------------------------- *
0500 MOVE 'Y1Y1=#TITLE,Y2Y2=#ABSTRACT,DB=#DATE,DA=#ORDER.' TO TRS.DSL-PARM
0510 *
0520 *
0530 CALL 'TRS' 'DSL' TRS.RC TRS.DSL-PARM
0540 *
0550 *
0560 IF TRS.RC NE 0
0570    MOVE TRS.RC TO TRS.RC1
0580    CALL 'TRS' 'CL' TRS.RC TRS.SAVE
0590    WRITE  'Error in TRS-DSL:' TRS.RC1
0600    STOP
0610 END-IF
0620 *
0630 SET KEY ALL
0640 SET KEY PF3 NAMED 'Quit'
0650 *
```

Display Menu and Invoke Selected Functions

A menu is displayed for selecting a desired function. The following functions are available:

A	Document Maintenance (store, update, delete);
R	Formatted Retrieval;
F	Freestyle Retrieval.

```
0660 * -------------------------------------------------------------------- *
0670 *  Display Menu                                                        *
0680 * -------------------------------------------------------------------- *
0690 REPEAT
0700  INPUT USING MAP 'TRS-MENU'
0710  IF *PF-KEY = 'PF3'
0720     ESCAPE BOTTOM
0730  END-IF
0740  DECIDE ON FIRST VALUE SEL
0750    VALUE 'A'
0760        FETCH 'TRS-ADD'
0770    VALUE 'R'
0780        FETCH 'TRS-QR'
0790    VALUE 'F'
0800        FETCH 'TRS-FQR'
0810    VALUE '.'
0820        ESCAPE BOTTOM
0830    NONE
0840        REINPUT 'Invalid function code.'
0850  END-DECIDE
0860 END-REPEAT
0870 *
```

Close the Adabas Text Retrieval Session

The CL call in line 0930 closes the Adabas Text Retrieval session.

```
0880 * -------------------------------------------------------------------- *
0890 *  Close TRS session                                                   *
0900 * -------------------------------------------------------------------- *
0910 *
0920 *
0930 CALL 'TRS' 'CL' TRS.RC TRS.SAVE
0940 *
0950 *
0960 SET CONTROL 'U'
0970 *
0980 END
```

Document Maintenance and Retrieval

TRS-ADD

The program TRS-ADD enables the user to perform the following functions.

- Select a document for update/delete;
- Store a document;
- Update a document;
- Invert a document (ADD call);
- Delete a document;
- Delete document index entries (DDS call).

Program Start Up

```
************************************************************************
*                                                                      *
*  ADABAS TEXT RETRIEVAL  Example Application                          *
*                                                                      *
*  Object   :  TRS-ADD                                                 *
*  Type     :  Program                                                 *
*  Function :  Store, update, delete and invert document               *
*  Author   :  Software AG                                             *
*                                                                      *
************************************************************************
*
DEFINE DATA LOCAL USING TRS-LDA          /* TRS Parameter
*
LOCAL
*
1 MAP1                                   /* Fields in Map
  2 ORDER    (A16)
  2 DATE     (N8)
  2 PRICE    (N3)
  2 TITLE    (A70)
  2 ABSTRACT (A70/12)
*
1 DOCUMENT VIEW OF TRS-DOCUMENT          /* Document View
  2 ORDER    (A16)
  2 DATE
  2 PRICE    (N3)
  2 TITLE
  2 ABSTRACT (12)
*
1 #ORDER-OLD (A16)                       /* Work Fields
1 #MSG       (A72)
*
END-DEFINE
*
```

Set Keys

```
* ------------------------------------------------------------------- *
* Set keys
* ------------------------------------------------------------------- *
SET KEY ALL
SET KEY PF2 NAMED 'Clear'
SET KEY PF3 NAMED 'Quit'
SET KEY PF4 NAMED 'Update'
SET KEY PF5 NAMED 'Delete'
SET KEY PF6 NAMED 'Store'
SET KEY PF8 NAMED 'Next '
SET KEY PF10 NAMED 'Query'
*
*
R1.                                    /* Main Loop
REPEAT
*
*
  INPUT WITH TEXT #MSG USING MAP 'TRS-ADDM'
*
```

Terminate

```
* ------------------------------------------------------------------- *
* Quit Function
* ------------------------------------------------------------------- *
  IF *PF-KEY = 'PF3'
    ESCAPE BOTTOM
  END-IF
*
```

Invoke Formatted Retrieval

```
* ------------------------------------------------------------------- *
* Invoke Query
* ------------------------------------------------------------------- *
  IF *PF-KEY = 'PF10'
    FETCH 'TRS-QR'
  END-IF
*
```

Clear Screen

```
* ------------------------------------------------------------------ *
* Clear screen
* ------------------------------------------------------------------ *
  IF *PF-KEY = 'PF2'
    RESET MAP1
    MOVE 'Input data and press enter.' TO #MSG
    ESCAPE TOP
  END-IF
*
```

Next Document

```
* ------------------------------------------------------------------ *
* Next document
* ------------------------------------------------------------------ *
  IF *PF-KEY = 'PF8'
    READ DOCUMENT BY ORDER = MAP1.ORDER
      IF MAP1.ORDER NE DOCUMENT.ORDER
        ESCAPE BOTTOM
      END-IF
    END-READ
    MOVE DOCUMENT.ORDER TO MAP1.ORDER
  END-IF
*
*
  IF MAP1.ORDER = ' '
    REINPUT 'Please enter Order Nr.'
  END-IF
*
```

Select Document for STORE/UPDATE/DELETE

```
* -------------------------------------------------------------------- *
* Select document
* -------------------------------------------------------------------- *
  IF *PF-KEY = 'ENTR' OR *PF-KEY = 'PF8'
    IF MAP1.ORDER = #ORDER-OLD
      ESCAPE TOP
    END-IF
*
    F1.
    FIND (1) DOCUMENT WITH ORDER = MAP1.ORDER
      MOVE BY NAME DOCUMENT TO MAP1
      MOVE MAP1.ORDER TO #ORDER-OLD
    END-FIND
*
    IF *NUMBER (F1.) > 0
      COMPRESS 'Order-Nr.:' MAP1.ORDER 'found.' INTO #MSG
    ELSE
      COMPRESS 'Order-Nr.:' MAP1.ORDER 'not found.' INTO #MSG
    END-IF
    ESCAPE TOP
*
  END-IF
*
```

Store Document

In this program segment, the document is stored in the document file and inverted by Adabas Text Retrieval. The document is inverted using the subroutine SR-INVERT specified in line 1300. This subroutine begins in line 2020.

```
01170 * -------------------------------------------------------------------- *
01180 * Store document
01190 * -------------------------------------------------------------------- *
01200   IF *PF-KEY = 'PF6'
01210      FIND (1) DOCUMENT WITH ORDER = MAP1.ORDER
01220        IF *NUMBER > 0
01230            COMPRESS 'Document' MAP1.ORDER 'already exists.' INTO #MSG
01240            REINPUT #MSG MARK *MAP1.ORDER
01250         END-IF
01260      END-FIND
01270      MOVE BY NAME MAP1 TO DOCUMENT
01280      STORE DOCUMENT
01290 *
01300      PERFORM SR-INVERT                   /* Invoke Inversion
01310 *
01320      END TRANSACTION
01330      COMPRESS 'Order-Nr.:' MAP1.ORDER 'successfully added.' INTO #MSG
01340      RESET #ORDER-OLD
01350   END-IF
01360 *
```

Update Document

The following program segment updates the document in the document file and re-inverts the document. The document is inverted using the subroutine SR-INVERT specified in line 1540. This subroutine begins in line 2020.

```
01370 * -------------------------------------------------------------------- *
01380 * Update document
01390 * -------------------------------------------------------------------- *
01400   IF *PF-KEY = 'PF4'
01410     IF MAP1.ORDER NE #ORDER-OLD
01420       REINPUT 'No change of Order-Nr. allowed. for update'
01430         MARK *MAP1.ORDER
01440     END-IF
01450     IF MAP1.DATE NE MASK(YYYYMMDD)
01460       REINPUT 'Please correct date.' MARK *MAP1.DATE
01470     END-IF
01480     FIND (1) DOCUMENT WITH ORDER = MAP1.ORDER
01490       MOVE BY NAME MAP1 TO DOCUMENT
01500 *
01510       UPDATE                             /* Invoke Inversion
01520 *
01530       RESET #ORDER-OLD
01540       PERFORM SR-INVERT
01550       END OF TRANSACTION
01560       COMPRESS 'Order-Nr.:' MAP1.ORDER 'successfully updated' INTO #MSG
01570     END-FIND
01580 *
01590   END-IF
01600 *
```

Delete Document

The documents are deleted from the document file. The selection of the relevant input entries is carried out in subroutine SR-DELETE specified in line 1830. This subroutine begins in line 2570.

```
01610 * ------------------------------------------------------------------- *
01620 * Delete document
01630 * ------------------------------------------------------------------- *
01640   IF *PF-KEY = 'PF5'
01650     F2.
01660     FIND DOCUMENT ORDER = MAP1.ORDER
01670       SET CONTROL 'WFL70C7B10/10'
01680       INPUT 'PLease retype Order-Nr.:'  #ORDER-OLD (AD=T'_')
01690       SET CONTROL 'WB'
01700       IF #ORDER-OLD NE MAP1.ORDER
01710         MOVE 'No record deleted.' TO #MSG
01720         ESCAPE BOTTOM
01730       END-IF
01740       DELETE
01750     END-FIND
01760 *
01770     IF *NUMBER (F2.) = 0
01780       BACKOUT TRANSACTION
01790       COMPRESS 'Document' MAP1.ORDER 'doesn"t exists.' INTO #MSG
01800       REINPUT #MSG MARK *MAP1.ORDER
01810     END-IF
01820 *
01830     PERFORM SR-DELETE                    /* Invoke Delete Index
01840 *
01850     RESET MAP1
01860     END OF TRANSACTION
01870     COMPRESS 'Order-Nr.:' MAP1.ORDER 'successfully deleted.' INTO #MSG
01880     RESET #ORDER-OLD
01890     ESCAPE TOP
01900   END-IF
01910 *
01920 *
01930 END-REPEAT
```

Invert Document

The SR-INVERT subroutine inverts the contents of the two free-text chapters TITLE and ABSTRACT.

Before the inversion by the ADD call, the TEXT parameter must be set to the name of the Adabas hyperdescriptor (Y1) which represents the free-text chapter TITLE in the document file. This is carried out by the DYP call in line 2120..

The ADD call is then executed in line 2180 which performs the inversion. Within the ADD call:

- The field MAP1.ORDER contains the current value of the document ID;
- The field TRS.ALEN contains the length of the text to be inverted, in this case 72 bytes;
- The field MAP1.TITLE(1) contains the one and only text line of the free-text chapter TITLE;
- The field TRS.DISN will contain the ISN assigned to the DFNR record by TRS;
- The constant, LAST, indicates that there are no subsequent parts of the free-text chapter to be inverted.

The procedure for inverting the chapter ABSTRACT is identical to that for TITLE above (see lines 2370 and 2430).

Bearing in mind that the name of the Adabas hyperdescriptor for ABSTRACT is Y2, the possible length of the free-text can be up to 864 bytes.

```
01940 ************************************************************************
01950 ************************************************************************
01960 ***  S u b r o u t i n e s  ********************************************
01970 ************************************************************************
01980 ************************************************************************
01990 *
02000 *
02010 ************************************************************************
02020 DEFINE SUBROUTINE SR-INVERT     /* Inversion Process for Document
02030 ************************************************************************
02040 *
02050 * ------------------------------------------------------------------- *
02060 * TRS-ADD    =====   Inversion for Document Chapter - TITLE
02070 * ------------------------------------------------------------------- *
02080 *
02090   MOVE 'TEXT=Y1Y1.' TO TRS.DYP-PARM
```

```
02100 *
02110 *
02120   CALL 'TRS' 'DYP' TRS.RC TRS.DYP-PARM
02130 *
02140 *
02150   MOVE 68 TO TRS.ALEN
02160 *
02170 *
02180   CALL 'TRS' 'ADD' TRS.RC MAP1.ORDER TRS.ALEN MAP1.TITLE
02190                    TRS.DISN 'LAST  '
02200 *
02210 *
02220   IF TRS.RC NE 0
02230     MOVE TRS.RC TO TRS.RC1
02240     BACKOUT TRANSACTION
02250     COMPRESS 'Error in TRS-ADD (Title) =>' TRS.RC1 INTO #MSG
02260     REINPUT #MSG
02270   END-IF
02280 *
02290 *
02300 * -------------------------------------------------------------------- *
02310 * TRS-ADD    =====   Inversion for Document Chapter - ABSTRACT
02320 * -------------------------------------------------------------------- *
02330 *
02340   MOVE 'TEXT=Y2Y2.' TO TRS.DYP-PARM
02350 *
02360 *
02370   CALL 'TRS' 'DYP' TRS.RC TRS.DYP-PARM
02380 *
02390 *
02400   MOVE 816 TO TRS.ALEN
02410 *
02420 *
02430   CALL 'TRS' 'ADD' TRS.RC MAP1.ORDER TRS.ALEN MAP1.ABSTRACT(1)
02440                    TRS.DISN 'LAST  '
02450 *
02460 *
02470   IF TRS.RC NE 0
02480     MOVE TRS.RC TO TRS.RC1
02490     BACKOUT TRANSACTION
02500     COMPRESS 'Error in TRS-ADD (Abstract) =>' TRS.RC1 INTO #MSG
02510     REINPUT #MSG
02520   END-IF
02530 *
02540 END-SUBROUTINE
02550 *
02560 *
```

Delete Document Index Entries

Within the SR-DELETE subroutine, the DDS call is invoked in order to remove index entries from the document index file (DSFNR).

The index entries for the free-text chapters TITLE and ABSTRACT must be deleted separately.

The deletion process takes place for ABSTRACT in lines 2670 and 2700 and for TITLE in lines 2860 and 2890. Prior to the invocation of the DDS call, the TEXT parameter must be set to the name of the Adabas hyperdescriptor representing the free-text chapter in question (see lines 2640 and 2830).

The field MAP1.ORDER contains the value of the document ID.

```
02570 ***********************************************************************
02580 DEFINE SUBROUTINE SR-DELETE     /* Delete Document Index
02590 ***********************************************************************
02600 *
02610 * ------------------------------------------------------------------- *
02620 * TRS-DDS    =====   Delete Index for Document Chapter - TITLE
02630 * ------------------------------------------------------------------- *
02640   MOVE 'TEXT=Y1Y1.' TO TRS.DYP-PARM
02650 *
02660 *
02670   CALL 'TRS' 'DYP' TRS.RC TRS.DYP-PARM
02680 *
02690 *
02700   CALL 'TRS' 'DDS' TRS.RC MAP1.ORDER 'SUM'
02710 *
02720 *
02730   IF TRS.RC NE 0
02740      MOVE TRS.RC TO TRS.RC1
02750      BACKOUT TRANSACTION
02760      COMPRESS 'Error in TRS-DDS (Title) =>' TRS.RC1 INTO #MSG
02770      REINPUT #MSG
02780   END-IF
02790 *
```

```
* ------------------------------------------------------------------- *
* TRS-DDS    =====   Delete Index for Document Chapter - ABSTRACT
* ------------------------------------------------------------------- *
  MOVE 'TEXT=Y2Y2.' TO TRS.DYP-PARM
*
*
  CALL 'TRS' 'DYP' TRS.RC TRS.DYP-PARM
*
*
  CALL 'TRS' 'DDS' TRS.RC MAP1.ORDER 'SUM'
*
*
  IF TRS.RC NE 0
     MOVE TRS.RC TO TRS.RC1
     BACKOUT TRANSACTION
     COMPRESS 'Error in TRS-DDS (Abstract) =>' TRS.RC1 INTO #MSG
     REINPUT #MSG
  END-IF
*
END-SUBROUTINE
*
*
*
*
FETCH 'MENU'
END
```

Formatted Retrieval

TRS-QR

The TRS-QR program represents the main retrieval part of the demonstration application. The program enables the user to enter specific queries for each of the categories in the document. The program returns the respective results and totals them by linking all relevant queries with the boolean operator “AND.” The program is broken up into the following subsections:

- Execute Query for the categories Order, Title, Abstract, Date (QR call);
- Produce Final Result (QR call);
- Invoke Overview.

Program Start Up

```
 *************************************************************************
 *                                                                       *
 *  ADABAS TEXT RETRIEVAL  Example Application                           *
 *                                                                       *
 *  Object   :  TRS-QR                                                   *
 *  Type     :  Program                                                  *
 *  Function :  Retrieval                                                *
 *  Author   :  Software AG                                              *
 *                                                                       *
 *************************************************************************
*
DEFINE DATA LOCAL USING TRS-LDA          /* TRS Parameter
*
LOCAL
*
1 #MAP                                   /* Fields in Map
  2 #ORDER(A60)
  2 #ORDER-R(N7)
  2 #TITLE(A60)
  2 #TITLE-R(N7)
  2 #ABSTRACT(A60)
  2 #ABSTRACT-R(N7)
  2 #DATE(A60)
  2 #DATE-R(N7)
  2 #RESULT(N7)
*
1 #MSG(A72)
*
END-DEFINE
*
```

Set Keys

```
0310* ------------------------------------------------------------------- *
0320*  Set keys
0330* ------------------------------------------------------------------- *
0340SET KEY ALL
0350SET KEY PF2 NAMED 'Clear'
0360SET KEY PF3 NAMED 'Quit'
0370SET KEY PF6 NAMED 'Over'
0380SET KEY PF10 NAMED 'Add'
0390*
0400REPEAT
0410*
0420  INPUT USING MAP 'TRS-QRM'
0430*
```

Terminate

```
0440* ------------------------------------------------------------------- *
0450*  Escape to menu
0460* ------------------------------------------------------------------- *
0470  IF *PF-KEY = 'PF3'
0480    ESCAPE BOTTOM
0490  END-IF
0500*
```

Invoke Document Processing

```
0510* ------------------------------------------------------------------- *
0520*  Invoke Add
0530* ------------------------------------------------------------------- *
0540  IF *PF-KEY = 'PF10'
0550    FETCH 'TRS-ADD'
0560  END-IF
0570*
```

Clear Screen

Clear Screen and release queries.

For the release of the queries a RQR call is used (see lines 0620 and 0650).

```
0580* ------------------------------------------------------------------- *
0590*  Clear the screen
0600* ------------------------------------------------------------------- *
0610  IF *PF-KEY = 'PF2'
0620*
0630*
0640    CALL 'TRS' 'RQR' TRS.RC 'DOCS0002'
0650*
0660*
0670    CALL 'TRS' 'RQR' TRS.RC 'DOCS0003'
0680*
0690*
0700    RESET #MAP
0710    ESCAPE TOP
0720  END-IF
0730*
```

Execute Query for ORDER

```
0740* -------------------------------------------------------------------- *
0750*  TRS Query   =====    ORDER-NUMBER
0760* -------------------------------------------------------------------- *
0770  RESET TRS.QUERY-G #ORDER-R #TITLE-R #ABSTRACT-R #DATE-R #RESULT
0780*
0790  IF #ORDER NOT EQ ' '
0800*
0810    MOVE 'DOCS0001' TO TRS.NAME
0820    COMPRESS '#ORDER' #ORDER TO TRS.QUERY
0830*
0840*
0850    CALL 'TRS' 'QR' TRS.RC TRS.QUERY TRS.QLEN TRS.NAME TRS.DERR TRS.LERR
0860                    TRS.MODE TRS.CID TRS.QTY TRS.TYPE
0870*
0880*
0890    IF TRS.RC NE 0
0900      MOVE TRS.RC TO TRS.RC1
0910      COMPRESS 'Error in TRS.QR =>' TRS.RC1 TO #MSG
0920      REINPUT #MSG MARK *#ORDER
0930    END-IF
0940*
0950    MOVE TRS.QTY TO #ORDER-R
0960    MOVE '#1' TO TRS.QUERY-G
0970*
0980  END-IF
0990*
1000*
```

Execute Query for TITLE

The query entered by the user for the formatted field TITLE is executed by the QR call in line 1100).

The search label entered by the user and the query expression are combined with the parameter TRS.QUERY for the QR call (see lines 1060-1070). The constant ’DOCS0002’ constitutes the query name in TRS.NAME. This will be used to reference back to this specific query later on in the program.

In the QR call:

- The TRS.QLEN parameter indicates the length of the query. In the example, it is 80 bytes;
- The parameters TRS.DERR and TRS.LERR contains information on syntax errors detected in the query;
- The constant “D” is chosen for the TRS.MODE parameter in order to indicate that a document retrieval must be carried out;
- The TRS.CID parameter contains the Adabas command ID which identifies the Adabas ISN list resulting from the query;
- The TRS.QTY parameter contains the number of selected documents. This parameter is shown to the user as the first result;
- The constant value ’=’, in the TRS.TYPE parameter, chooses the default selection mode to be PRECISE.

In the field TRS.QUERY-G of line 1220, the constant entry ’#2’ indicates that the result of this query will be incorporated into the final query (for determining the final result).

```
1010* -------------------------------------------------------------------- *
1020*  TRS Query   =====    TITLE
1030* -------------------------------------------------------------------- *
1040  IF #TITLE NOT EQ ' '
1050*
1060    MOVE 'DOCS0002' TO TRS.NAME
1070    COMPRESS '#TITLE' #TITLE TO TRS.QUERY
1080*
1090*
1100    CALL 'TRS' 'QR' TRS.RC TRS.QUERY TRS.QLEN TRS.NAME TRS.DERR TRS.LERR
1110                    TRS.MODE TRS.CID TRS.QTY TRS.TYPE
1120*
1130*
1140    IF TRS.RC NE 0
1150      MOVE TRS.RC TO TRS.RC1
1160      COMPRESS 'Error in TRS.QR =>' TRS.RC1 TO #MSG
1170      REINPUT #MSG MARK *#TITLE
1180    END-IF
1190*
1200    MOVE TRS.QTY TO #TITLE-R
1210    IF TRS.QUERY-G EQ ' '
1220      MOVE '#2' TO TRS.QUERY-G
1230    ELSE
1240      COMPRESS TRS.QUERY-G 'AND #2' TO TRS.QUERY-G
1250    END-IF
1260*
1270   ELSE
1280*
1290*
1300    CALL 'TRS' 'RQR' TRS.RC 'DOCS0002'
1310*
1320*
1330  END-IF
1340*
1350*
```

Execute Query for ABSTRACT and DATE

A QR call is used to select documents according to the queries entered by the user for the fields ABSTRACT and DATE. Note that the QR call in lines 1450 and 1460 is identical to that used for TITLE above.

```
1360* -------------------------------------------------------------------- *
1370*  TRS Query   =====    ABSTRACT
1380* -------------------------------------------------------------------- *
1390  IF #ABSTRACT NOT EQ ' '
1400*
1410    MOVE 'DOCS0003' TO TRS.NAME
1420    COMPRESS '#ABSTRACT' #ABSTRACT TO TRS.QUERY
1430*
1440*
1450    CALL 'TRS' 'QR' TRS.RC TRS.QUERY TRS.QLEN TRS.NAME TRS.DERR TRS.LERR
1460                    TRS.MODE TRS.CID TRS.QTY TRS.TYPE
1470*
1480*
1490    IF TRS.RC NE 0
1500      MOVE TRS.RC TO TRS.RC1
1510      COMPRESS 'Error in TRS.QR =>' TRS.RC1 TO #MSG
1520      REINPUT #MSG MARK *#ABSTRACT
1530    END-IF
1540*
1550    MOVE TRS.QTY TO #ABSTRACT-R
1560    IF TRS.QUERY-G EQ ' '
1570      MOVE '#3' TO TRS.QUERY-G
1580    ELSE
1590      COMPRESS TRS.QUERY-G 'AND #3' TO TRS.QUERY-G
1600    END-IF
1610*
1620   ELSE
1630*
1640*
1650    CALL 'TRS' 'RQR' TRS.RC 'DOCS0003'
1660*
1670*
1680  END-IF
1690*
1700*
1710* -------------------------------------------------------------------- *
1720*  TRS Query   =====    DATE
1730* -------------------------------------------------------------------- *
1740  IF #DATE NOT EQ ' '
```

```
*
  MOVE 'DOCS0004' TO TRS.NAME
  COMPRESS '#DATE' #DATE TO TRS.QUERY
*
*
  CALL 'TRS' 'QR' TRS.RC TRS.QUERY TRS.QLEN TRS.NAME TRS.DERR TRS.LERR
                  TRS.MODE TRS.CID TRS.QTY TRS.TYPE
*
*
  IF TRS.RC NE 0
    MOVE TRS.RC TO TRS.RC1
    COMPRESS 'Error in TRS.QR =>' TRS.RC1 TO #MSG
    REINPUT #MSG MARK *#DATE
  END-IF
*
  MOVE TRS.QTY TO #DATE-R
  IF TRS.QUERY-G EQ ' '
    MOVE '#4' TO TRS.QUERY-G
  ELSE
    COMPRESS TRS.QUERY-G 'AND #4' TO TRS.QUERY-G
  END-IF
*
END-IF
*
*
```

Produce Final Result

A QR call is executed in line 2090 to produce the final total result based on the queries which have already been executed.

This final result is stored under the name DOCS0011 in line 2050.

```
2000* -------------------------------------------------------------------- *
2010*  TRS Query   =====    Total
2020* -------------------------------------------------------------------- *
2030  IF TRS.QUERY-G NOT EQ ' '
2040*
2050    MOVE 'DOCS0011'  TO TRS.NAME
2060    MOVE TRS.QUERY-G TO TRS.QUERY
2070*
2080*
2090    CALL 'TRS' 'QR' TRS.RC TRS.QUERY TRS.QLEN TRS.NAME TRS.DERR TRS.LERR
2100                    TRS.MODE TRS.CID TRS.QTY TRS.TYPE
2110*
2120*
2130    IF TRS.RC NE 0
2140      MOVE TRS.RC TO TRS.RC1
2150      COMPRESS 'Error in TRS.QR =>' TRS.RC1 TO #MSG
2160      REINPUT #MSG
2170    END-IF
2180*
2190    MOVE TRS.QTY TO #RESULT
2200*
2210  END-IF
2220*
```

Invoke Overview

The program TRS-EIS is invoked to create an overview of all documents selected by the final query.

```
2230* ---------------------------------------------------------------------- *
2240*  Invoke Overview
2250* ---------------------------------------------------------------------- *
2260  IF *PF-KEY = 'PF6'
2270    IF #RESULT = 0
2280      REINPUT 'No final result build.'
2290    ELSE
2300      FETCH RETURN 'TRS-EIS'
2310    END-IF
2320  END-IF
2330*
2340END-REPEAT
2350*
2360FETCH 'MENU'
2370*
2380*
2390END
```

Overview of Selected Documents

TRS-EIS

The program TRS-EIS creates an overview of the selected documents using the EISS, EISG and EISE calls. The user can then select documents to be displayed by marking them with “X”. The program is divided into the following subsections:

- Resume Query (QR call);
- Create Overview (EISG call);
- Page Up;
- Page Down;
- Select Item for Display.

Program Start Up

```
************************************************************************
*                                                                      *
*  ADABAS TEXT RETRIEVAL  Example Application                          *
*                                                                      *
*  Object   :  TRS-EIS                                                 *
*  Type     :  Program                                                 *
*  Function :  Retrieval overview                                      *
*  Author   :  Software AG                                             *
*                                                                      *
************************************************************************
*
DEFINE DATA LOCAL USING TRS-LDA          /* TRS-Parameter
*
LOCAL
*
 1 DOCUMENT VIEW OF TRS-DOCUMENT         /* Document View
   2 ORDER
   2 DATE
   2 TITLE
*
 1 MAP1                                  /* Fields in Map
   2 LINE(A72/16)
   2 MARK(A1/16)
   2 MARK-CV(C/16)
*
1 MSG(A72)
*
1 J(N2)                                  /* Work Fields
1 K(N2)
1 I(N7)
1 P-FROM(N7)
1 P-THRU(N7)
1 ORD(A16/16)
*
1 LINE1(A72)
1 REDEFINE LINE1
  2 ORDER(A11)
  2 FILLER1 (A1)
  2 DATE(N8)
  2 FILLER2 (A1)
  2 LINE-TEXT(A51)
*
END-DEFINE
*
```

Set Keys

```
0450* ------------------------------------------------------------------ *
0460*  Set keys
0470* ------------------------------------------------------------------ *
0480SET KEY ALL
0490SET KEY PF3 NAMED 'Quit'
0500SET KEY PF6 NAMED 'Disp'
0510SET KEY PF7 NAMED 'Back'
0520SET KEY PF8 NAMED 'For'
0530*
```

Resume Query

A QR call in line 0610 is executed in order to resume the result of the final query by the program TRS-QR and to sort the selected documents according to the formatted field ORDER (see line 0570).

The EISS call in line 0740 is used to start browsing through an ISN set created by a QR call.

```
0540* ------------------------------------------------------------------ *
0550*  Resume Queries
0560* ------------------------------------------------------------------ *
0570MOVE '#11 SORT #ORDER' TO TRS.QUERY
0580MOVE 'DOCS0012' TO TRS.NAME
0590*
0600*
0610CALL 'TRS' 'QR' TRS.RC TRS.QUERY TRS.QLEN TRS.NAME TRS.DERR TRS.LERR
0620                TRS.MODE TRS.CID TRS.QTY TRS.TYPE
0630*
0640*
0650IF TRS.RC NE 0
0660   MOVE TRS.RC TO TRS.RC1
0670   WRITE 'INTERNAL ERROR'
0680   STOP
0690END-IF
0700*
0710MOVE TRS.QTY TO TRS.QTY1
0720*
0730*
0740CALL 'TRS' 'EISS' TRS.RC TRS.CID TRS.TYPE TRS.QTY
0750*
0760*
0770MOVE 1 TO P-FROM
0780*
```

Create Overview

An overview is of the selected documents is created. These documents are fetched via ISNs which have been provided by the EISG call in line 1000. This call contains the following parameters:

- The TRS.CID parameter contains the Adabas Command ID assigned to the query previously executed.
- The constant “D” is assigned to the parameter TRS.TYPE to indicate that a document selection was executed.
- The TRS.QTY parameter contains the number of documents selected by the previous QR call;
- The TRS.POS parameter must contain the relative position of the requested ISN inside the Adabas ISN set. In this case, the user variable “I” is used to browse through the selected documents;
- The parameter TRS.ISN will contain the Adabas ISN selected by the EISG call.

The document represented by the ISN provided by the EISG call is then accessed by a GET command in line 1050.

```
0790* -------------------------------------------------------------------- *
0800*  Create Overview
0810* -------------------------------------------------------------------- *
0820REPEAT
0830*
0840                                 /* Calculate Position in Set
0850  MOVE P-FROM TO P-THRU
0860  ADD 15 TO P-THRU
0870  IF P-THRU GT TRS.QTY1
0880     MOVE TRS.QTY1 TO P-THRU
0890  END-IF
0900*
0910  RESET MAP1 K
0920  MOVE (AD=NP) TO MARK-CV(*)
0930*
0940*
0950  FOR I = P-FROM TO P-THRU
0960*
0970    MOVE I TO TRS.POS
0980*
0990*
1000    CALL 'TRS' 'EISG' TRS.RC TRS.CID TRS.TYPE TRS.QTY TRS.POS TRS.ISN
1010*
1020*
1030    MOVE TRS.ISN TO TRS.ISN1
1040*
1050    GET DOCUMENT TRS.ISN1
1060*
1070        ADD 1 TO K
1080        MOVE DOCUMENT.ORDER TO LINE1.ORDER
1090        MOVE DOCUMENT.ORDER TO ORD(K)
1100        MOVE DOCUMENT.DATE  TO LINE1.DATE
1110        MOVE TITLE          TO LINE-TEXT
1120        MOVE LINE1 TO LINE(K)
1130        RESET MARK-CV(K)
1140*
1150  END-FOR
1160*
1170*
1180  INPUT WITH TEXT MSG USING MAP 'TRS-EISM'
1190*
1200*
1210  RESET MSG
1220*
```

Escape

```
1230* -------------------------------------------------------------------- *
1240*  Escape
1250* -------------------------------------------------------------------- *
1260  IF *PF-KEY = 'PF3'
1270     ESCAPE BOTTOM
1280  END-IF
1290*
```

Page Up

Scroll back through the set of documents.

```
1300* -------------------------------------------------------------------- *
1310*  Page-Up
1320* -------------------------------------------------------------------- *
1330  IF *PF-KEY = 'PF7'
1340     SUBTRACT 16 FROM P-FROM
1350     IF P-FROM LT 1
1360        MOVE 'This is the first page.' TO MSG
1370       MOVE 1 TO P-FROM
1380     END-IF
1390  END-IF
1400*
```

Page Down

Scroll forward through the set of documents.

```
1410* -------------------------------------------------------------------- *
1420*  Page-Down
1430* -------------------------------------------------------------------- *
1440  IF *PF-KEY = 'PF8'
1450     ADD 16 TO P-FROM
1460     IF P-FROM GT TRS.QTY1
1470       MOVE 'This is the last page.' TO MSG
1480       SUBTRACT 16 FROM P-FROM
1490     END-IF
1500  END-IF
1510*
```

Select Item for Display

The subprogram TRS-DISP is invoked in line 1650 to display a document (see line 1560). The document-iol (ORDER) is passed as parameter to the subprogram.

At the end, an EISE call is issued in line 1960 to end browsing.

```
1520* -------------------------------------------------------------------- *
1530*  Select item for display
1540* -------------------------------------------------------------------- *
1550  IF *PF-KEY = 'ENTR' OR *PF-KEY = 'PF6'
1560*
1570     IF *PF-KEY = 'PF6'
1580        MOVE ALL 'X' TO MARK(1:K)
1590     END-IF
1600*
1610     FOR J = 1 TO 16
1620         IF MAP1.MARK (J) NE ' '
1630            MOVE LINE(J) TO LINE1
1640*
1650            CALLNAT 'TRS-DISP' ORD(J) MSG
1660*
1670            IF *PF-KEY = 'PF3'
1680               ESCAPE BOTTOM
1690            END-IF
1700*
1710            IF *PF-KEY = 'PF6'
1720               MOVE 'This is the first page.' TO MSG
1730               RESET J
1740            END-IF
1750*
1760            IF *PF-KEY = 'PF8' AND J = K
1770               MOVE 'This is the last page.' TO MSG
1780               SUBTRACT 1 FROM J
1790            END-IF
1800*
1810            IF *PF-KEY = 'PF7'
1820               SUBTRACT 2 FROM J
1830               IF J LT 0
1840                  MOVE 'This is the first page.' TO MSG
1850                  RESET J
1860               END-IF
1870            END-IF
1880*
1890         END-IF
```

```
1900     END-FOR
1910  END-IF
1920*
1930END-REPEAT
1940*
1950*
1960CALL 'TRS' 'EISE' TRS.RC TRS.CID TRS.TYPE
1970*
1980*
1990END
```

Document Display

TRS-DISP

The program TRS-DISP displays a single document selected from the document overview provided by the program TRS-EIS. The HIGH call is used to mark the words which fulfill the search criterion specified in the QR calls. The program is divided into the following major subsections:

- Find document;
- Highlight document (HIGH call);
- Display documents.

Program Start Up

```
************************************************************************
*                                                                      *
*  ADABAS TEXT RETRIEVAL  Example Application                          *
*                                                                      *
*  Object   :  TRS-DISP                                                *
*  Type     :  Subprogram                                              *
*  Function :  Display selected documents                              *
*  Author   :  Software AG                                             *
*                                                                      *
************************************************************************
*
DEFINE DATA PARAMETER
*
1 PARA
  2 ORDER (A16)
  2 MSG   (A72)
*
LOCAL USING TRS-LDA
LOCAL
*
1 MAP1
  2 ORDER(A16)
  2 PRICE(N3)
  2 DATE(N8)
  2 TITLE1(A70)
  2 ABSTRACT1(A70/12)
*
1 DOCUMENT VIEW OF TRS-DOCUMENT
  2 ORDER
  2 DATE
  2 PRICE
  2 TITLE
  2 C*ABSTRACT
  2 ABSTRACT (12)
1 I(N2)
*
END-DEFINE
*
REPEAT
*
  RESET MAP1
*
```

Find Document

This procedure collects the document information for display and highlighting.

```
0430* ------------------------------------------------------------------- *
0440* Find Document
0450* ------------------------------------------------------------------- *
0460  FIND DOCUMENT WITH ORDER = PARA.ORDER
0470*
0480    MOVE DOCUMENT.ORDER TO MAP1.ORDER
0490    MOVE DOCUMENT.PRICE TO MAP1.PRICE
0500    MOVE DOCUMENT.DATE TO MAP1.DATE
0510    MOVE 70 TO TRS.HLEN
0520    MOVE DOCUMENT.TITLE    TO TRS.HTEXT1
0530    MOVE 0  TO TRS.CURSOR
0540*
```

Highlight Document

A HIGH call is issued in order to highlight the term or terms specified in a query. Prior to highlighting, a DYP call (see line 0600 and 0880) must be executed in order to set the TEXT parameter to the name of the Adabas hyperdescriptor representing the free-text chapter in question.

In the HIGH call, the DOCUMENT.ORDER parameter contains the document ID. This process is also carried out for the document abstract (see lines 0830-1020).

The constant DOCS0002 is the name of the query (issued in the program TRS-QR) for which the highlighting is to be performed. The input parameter TRS.HTEXT contains the source text to be highlighted (in this case the title). By use of the Natural dynamic attribute parameter (DY), the TRS.HTEXT2 output parameter, which contains the source text including prefixes and suffixes created by the HIGH call, can be used for immediate physical highlighting.

The constants '<' and '>' represent the suffix and prefix to be used to mark the words fulfilling the search criteria. Highlighting is achieved using the Natural dynamic attribute facility.

The TRS.CURSOR parameter is an internal variable which controls the highlighting process. Prior to the start of the highlighting process, it must be set to zero. It must not be changed during intermediate processing (see line 0740).

```
0550*   ------------------------------------------------------------------ *
0560*   TRS High    =====   Highlight Document Chapter - TITLE
0570*   ------------------------------------------------------------------ *
0580*
0590*
0600    CALL 'TRS' 'DYP'  TRS.RC 'TEXT=Y1Y1.'
0610*
0620*
0630    CALL 'TRS' 'HIGH' TRS.RC DOCUMENT.ORDER 'DOCS0002'
0640                      TRS.HTEXT TRS.HTEXT2 TRS.HLEN '<' '>' TRS.CURSOR
0650*
0660*
0670    IF NOT (TRS.RC = 0 OR = 6)
0680      MOVE TRS.RC TO TRS.RC1
0690      INPUT(AD=OIL) 'ERROR IN TRS-HIGH :' TRS.RC1
0700      STOP
0710    END-IF
0720*
0730    MOVE TRS.HTEXT2 TO MAP1.TITLE1
0740    MOVE 0  TO TRS.CURSOR
0750*
0760    FOR I = 1 TO C*DOCUMENT.ABSTRACT
0770      IF I > 12
0780        ESCAPE BOTTOM
0790      END-IF
0800      MOVE 70 TO TRS.HLEN
0810      MOVE DOCUMENT.ABSTRACT(I) TO TRS.HTEXT1
0820*
0830*     ---------------------------------------------------------------- *
0840*     TRS High    =====   Highlight Document Chapter - ABSTRACT
0850*     ---------------------------------------------------------------- *
0860*
0870*
0880      CALL 'TRS' 'DYP'  TRS.RC 'TEXT=Y2Y2.'
0890*
0900*
0910      CALL 'TRS' 'HIGH' TRS.RC DOCUMENT.ORDER 'DOCS0003' TRS.HTEXT
0920                        TRS.HTEXT2 TRS.HLEN '<' '>' TRS.CURSOR
0930*
```

```
*
      IF NOT (TRS.RC = 0 OR = 6)
        MOVE TRS.RC TO TRS.RC1
        INPUT(AD=OIL) 'ERROR IN TRS-HIGH :' TRS.RC1
        STOP
      END-IF
      MOVE TRS.HTEXT2 TO MAP1.ABSTRACT1(I)
    END-FOR
  END-FIND
*
```

Display Documents

Display the documents with highlighting.

Note:
The Natural dynamic attribute feature is used to highlight the words marked by the HIGH call.

```
* -------------------------------------------------------------------- *
* Display Highlighted Document
* -------------------------------------------------------------------- *
  INPUT WITH TEXT MSG
        USING MAP 'TRS-DISM'
*
  IF *PF-KEY = 'PF3' OR *PF-KEY = 'PF6' OR
     *PF-KEY = 'PF7' OR *PF-KEY = 'PF8'
    RESET MSG
    ESCAPE ROUTINE
  END-IF
*
END-REPEAT
*
END
```

Index Display

TRS-HLP

The help routine TRS-HLP shows the values of the formatted fields and the free-text chapters. For formatted fields DATE and ORDER, a simple histogram is used. For the free-text chapters, logical reads on the “word fields” of the vocabulary and subsequent QR calls are used to obtain the number of documents where the words occur.

The program is divided into the following subsections:

- Display Available Values for ORDER;
- Display Available Values for DATE;
- Display Vocabulary for TITLE or ABSTRACT;
- Show Screen.

Program Start Up

```
************************************************************************
*                                                                      *
*  ADABAS TEXT RETRIEVAL  Example Application                          *
*                                                                      *
*  Object   :  TRS-HLP                                                 *
*  Type     :  Helproutine                                             *
*  Function :  Display category index                                  *
*  Author   :  Software AG                                             *
*                                                                      *
************************************************************************
*
DEFINE DATA PARAMETER
*
1 PARA
  2 FIELD (A65)
  2 VALUE (A60)
  2 REDEFINE VALUE
    3 DATE (N4)
*
LOCAL USING TRS-LDA
*
LOCAL
1 VOC VIEW OF TRS-VOCABULARY
  2 WORD
  2 ORIGINAL-WORD
1 DOC1 VIEW OF TRS-DOCUMENT
  2 ORDER
1 DOC2 VIEW OF TRS-DOCUMENT
  2 DATE
1 MAP1
  2 CAT  (A10)
  2 MARK (A1/10)
  2 WORD (A30/10)
  2 QTY  (N5/10)
  2 CV   (C/10)
*
1 #I (N2)
1 #J (N2)
1 #K (A1)
1 #D (N4)
1 #A (A60)
1 #V (A34)
END-DEFINE
*
SET   KEY PF3
MOVE (AD=PN) TO CV(*)
*
*
IF NOT(PARA.FIELD = '#TITLE' OR = '#ABSTRACT' )
*
```

Display Available Values for ORDER

A HISTOGRAM statement is used in line 550 to determine the values present for the formatted field ORDER.

```
0510* -------------------------------------------------------------------- *
0520*  Display available values for Order                                  *
0530* -------------------------------------------------------------------- *
0540  IF PARA.FIELD = '#ORDER'
0550    HISTOGRAM DOC1 FOR ORDER STARTING FROM PARA.VALUE
0560      ADD 1 TO #I
0570      MOVE DOC1.ORDER  TO MAP1.WORD(#I)
0580      MOVE *NUMBER     TO MAP1.QTY(#I)
0590      RESET CV(#I)
0600      IF #I GE 10
0610        PERFORM SR-SCREEN
0620        IF *PF-KEY = 'PF3'
0630          ESCAPE ROUTINE
0640        END-IF
0650      END-IF
0660    END-HISTOGRAM
0670  END-IF
```

Display Available Values for DATE

A HISTOGRAM statement is used in line 770 to determine the values present for the formatted field DATE.

```
0680* ------------------------------------------------------------------- *
0690*  Display available values for Date                                  *
0700* ------------------------------------------------------------------- *
0710  IF PARA.FIELD = '#DATE'
0720    IF PARA.DATE NE MASK(YYMM)
0730      MOVE PARA.DATE TO #D
0740    END-IF
0750    RESET PARA.VALUE
0760*
0770    HISTOGRAM DOC2 FOR DATE STARTING FROM #D
0780      ADD 1 TO #I
0790      MOVE DOC2.DATE  TO MAP1.WORD(#I)
0800      MOVE *NUMBER    TO MAP1.QTY(#I)
0810      RESET CV(#I)
0820      IF #I GE 10
0830        PERFORM SR-SCREEN
0840        IF *PF-KEY = 'PF3'
0850          ESCAPE ROUTINE
0860        END-IF
0870      END-IF
0880    END-HISTOGRAM
0890*
0900  END-IF
0910ELSE
0920*
```

Display Vocabulary for TITLE or ABSTRACT

The vocabulary file is read in line 1010 and a QR call is issued in line 1080 to determine how often the queried word is found in the contents of the relevant free-text chapter. If the number of documents selected by this query is greater than zero, the word becomes part of the display. The ‘‘original word’’ representing the non-standard word is used for the display to show the word in lower/upper case and special characters (see line 1190).

```
0930* -------------------------------------------------------------------- *
0940*  DISPLAY VOCABULARY FOR TITLE OR ABSTRACT                              *
0950* -------------------------------------------------------------------- *
0960  MOVE PARA.VALUE TO #A
0970  IF PARA.VALUE < H'81'
0980    MOVE H'81' TO #A
0990  END-IF
1000*
1010  READ VOC BY WORD = #A
1020*
1030    COMPRESS '''' VOC.WORD '''' TO #V LEAVING NO
1040    COMPRESS PARA.FIELD #V TO TRS.QUERY
1050    MOVE 'DOCS0020'              TO TRS.NAME
1060*
1070*
1080    CALL 'TRS' 'QR' TRS.RC TRS.QUERY TRS.QLEN TRS.NAME TRS.DERR TRS.LERR
1090                    TRS.MODE TRS.CID TRS.QTY TRS.TYPE
1100*
1110*
1120    IF TRS.RC NE 0
1130      MOVE TRS.RC TO TRS.RC1
1140      INPUT (AD=OIL) 'ERROR IN TRS.QR =>' TRS.RC1
1150    END-IF
1160*
1170    IF TRS.QTY > 0
1180      ADD 1 TO #I
1190      MOVE VOC.ORIGINAL-WORD TO MAP1.WORD(#I)
1200      MOVE TRS.QTY TO MAP1.QTY(#I)
1210      RESET CV(#I)
1220    END-IF
1230*
1240    IF #I GE 10
1250      PERFORM SR-SCREEN
1260      IF *PF-KEY = 'PF3'
1270        ESCAPE ROUTINE
1280      END-IF
1290    END-IF
1300  END-READ
1310END-IF
1320IF #I > 0
1330  PERFORM SR-SCREEN
1340END-IF
1350*
```

Show Screen

The selected values are displayed to screen and can be selected for inclusion in queries.

```
1360************************************************************************
1370DEFINE SUBROUTINE SR-SCREEN    /* Display Screen
1380************************************************************************
1390*
1400MOVE PARA.FIELD TO MAP1.CAT
1410*
1420INPUT USING MAP 'TRS-HLPM'
1430*
1440IF *PF-KEY = 'PF3'
1450  ESCAPE ROUTINE
1460END-IF
1470*
1480* -------------------------------------------------------------------- *
1490* Check if words are marked
1500* -------------------------------------------------------------------- *
1510FOR #J = 1 TO #I
1520  IF MAP1.MARK(#J) NE ' '
1530    IF PARA.VALUE NE ' '
1540      MOVE ',' TO #K
1550    END-IF
1560    COMPRESS PARA.VALUE #K MAP1.WORD(#J) INTO PARA.VALUE LEAVING NO
1570    IF NOT (PARA.FIELD = '#TITLE' OR = '#ABSTRACT') AND
1580      PARA.VALUE NE ' '
1590      ESCAPE BOTTOM
1600    END-IF
1610  END-IF
1620END-FOR
1630*
1640RESET MAP1 #I #J
1650MOVE (AD=PN) TO CV(*)
1660*
1670END-SUBROUTINE
1680END
```

Freestyle Retrieval

TRS-FQR

The program TRS-FQR enables the user to enter queries which conform to Adabas Text Retrieval query syntax. The program is divided into the following logical units.

- Delete queries (RQR);
- Copy query;
- Create Natural retained set (RET call);
- Execute query (QR call).

Program Start Up

```
************************************************************************
*                                                                      *
*  ADABAS TEXT RETRIEVAL  Example Application                          *
*                                                                      *
*  Object   :  TRS-FQR                                                 *
*  Type     :  Program                                                 *
*  Function :  Freestyle retrieval                                     *
*  Author   :  Software AG                                             *
*                                                                      *
************************************************************************
*
DEFINE DATA LOCAL USING TRS-LDA
*
LOCAL
1 #CHAPTER(A10)
1 #QUERY   (A62)
1 #NR (N2/10)
1 #MARK(A1/10)
1 #RESULT(N5/10)
1 #PQR(A62/10)
1 #MSG   (A60)
1 #I     (N4)
1 #J     (N4)
1 #SETID (A32) INIT<'TRSSET'>
1 #QNAM  (A8)
1 REDEFINE #QNAM
 2 HEADER (A4)
 2 COUNT  (N4)
1 CV1    (C/10)
*
END-DEFINE
*
```

Set Keys

```
0330* -------------------------------------------------------------------- *
0340*  Set keys
0350* -------------------------------------------------------------------- *
0360SET KEY ALL
0370SET KEY PF2 NAMED 'Delete'
0380SET KEY PF4 NAMED 'Copy '
0390SET KEY PF3 NAMED 'Quit'
0400SET KEY PF6 NAMED 'Over'
0410*
0420MOVE (AD=PN) TO CV1(*)
0430*
0440REPEAT
0450*
0460  RESET #MARK(*)
0470*
0480  INPUT WITH TEXT #MSG USING MAP 'TRS-FQRM'
0490*
```

Terminate

```
0500* -------------------------------------------------------------------- *
0510*  Escape to menu
0520* -------------------------------------------------------------------- *
0530  IF *PF-KEY = 'PF3'
0540    ESCAPE BOTTOM
0550  END-IF
0560*
```

Delete Queries

The user deletes previously executed queries by marking them with “D” and pressing PF2. The queries are released by executing a RQR call as seen in line 0660.

```
0570* -------------------------------------------------------------------- *
0580*  Delete Queries
0590* -------------------------------------------------------------------- *
0600  IF *PF-KEY = 'PF2'
0610    FOR #I = 1 TO 10
0620      IF #MARK(#I) EQ  'D'
0630         MOVE #I TO #QNAM.COUNT
0640*
0650*
0660        CALL 'TRS' 'RQR' TRS.RC #QNAM
0670*
0680*
0690        RESET #RESULT(#I) #NR(#I) #PQR(#I) #MARK(#I)
0700        MOVE (AD=PN) TO CV1(#I)
0710      END-IF
0720    END-FOR
0730    ESCAPE TOP
0740  END-IF
0750*
```

Copy Query

A previously executed query can be copied to make up a new query which can be modified and executed.

```
0760* -------------------------------------------------------------------- *
0770*  Copy Query
0780* -------------------------------------------------------------------- *
0790  IF *PF-KEY = 'PF4'
0800    FOR #I = 1 TO 10
0810      IF #MARK(#I) EQ  'C'
0820        MOVE #PQR(#I) TO #QUERY
0830        RESET #MARK (*)
0840        ESCAPE BOTTOM
0850      END-IF
0860    END-FOR
0870    ESCAPE TOP
0880  END-IF
0890*
```

Execute Query

A QR call (see line 1400) is executed to perform the query. The query and its results are put on the stack of executed queries.

```
1170* -------------------------------------------------------------------- *
1180*  TRS Queries
1190* -------------------------------------------------------------------- *
1200  IF #QUERY = ' '
1210     REINPUT 'No query specified.'
1220  END-IF
1230*
1240  FOR #I = 1 TO 10
1250      IF #PQR(#I) = ' '
1260         ESCAPE BOTTOM   /* Check for empty slot
1270      END-IF
1280  END-FOR
1290  IF #I = 11
1300    REINPUT
1310      'Stack is full ! delete queries by marking with "D" and PF2'
1320  END-IF
1330*
1340  MOVE 'DOCS'    TO #QNAM.HEADER
1350  MOVE #I        TO #QNAM.COUNT
1360  MOVE #QNAM     TO TRS.NAME
1370  COMPRESS #CHAPTER #QUERY TO TRS.QUERY
1380*
1390*
1400  CALL 'TRS' 'QR' TRS.RC TRS.QUERY TRS.QLEN TRS.NAME TRS.DERR TRS.LERR
1410                  TRS.MODE TRS.CID TRS.QTY TRS.TYPE
1420*
1430*
1440  MOVE (AD=I)    TO CV1(#I)
1450  MOVE TRS.QUERY TO #PQR(#I)
1460  MOVE TRS.QTY   TO #RESULT(#I)
1470  MOVE #I        TO #NR(#I)
1480*
1490  IF TRS.RC NE 0
1500    MOVE TRS.RC TO TRS.RC1
1510    COMPRESS 'Error in TRS.QR =>' TRS.RC1 TO #MSG
1520  END-IF
1530  RESET  #QUERY
1540*
1550 END-REPEAT
1560*
1570 FETCH 'MENU'
1580 END
```

APPENDIX A — MESSAGES AND CODES

General Return Codes

Return Code 1
Explanation: Invalid number of parameters.

Return Code 2
Explanation: Storage allocation failed – increase commen area size in ”BC” call.

Return Code 3
Explanation: No common area allocated. Use the BC call to allocate it.

Return Code 4
Explanation: Internal buffer not found or end of line in scanner reached.

Return Code 5
Explanation: EISG error in browsing. Check EIS* function parameters.

Return Code 6
Explanation: Referenced query not found.

Return Code 7
Explanation: Invalid request for vocabulary query.

Return Code 8
Explanation: User table overflow.

Return Code 9
Explanation: Invalid use of proximity search.

Return Code 10
Explanation: Middle word search and no V2 field.

Return Code 11
Explanation: Too many terms in aspect operation.

Return Code 12
Explanation: Phonetic search and no V6 field.

Return Code 13
Explanation: Number of words in document exceeds initial estimation.

Return Code 14
Explanation: Invalid function call.

Return Code 15
Explanation: Common area allocation failed.

Return Code 16
Explanation: Invalid use of search lables.

Return Code 17
Explanation: Record set unexpectedly empty.

Return Code 18
Explanation: Invalid parameter in ”QR” (query type ”D” or ”V”).

Return Code 19
Explanation: Syntax error detected in cataloging rules.

Return Code 20
Explanation: Vocabulary isn limit reached during query (MAXVSET).

Return Code 21
Explanation: Invalid parameter. Only ”LAST” or ”NOLAST” allowed.

Return Code 22
Explanation: Invalid parameter. Only ”ALL” or ”SUM” allowed.

Return Code 23
Explanation: Invalid default mode parameter.

Return Code 24
Explanation: Second ”BC” and common area size is different.

Return Code 25
Explanation: No document record in DFNR file.

Return Code 26
Explanation: Invalid parameter. Only ”SUM” allowed.

Return Code 27
Explanation: Unable to load user exit.

Return Code 28
Explanation: Unable to locate user exit’s entry point.

Return Code 30
Explanation: Invalid isn in batch load.

Return Code 31
Explanation: FDT area not found.

Return Code 37
Explanation: Invalid hyper descriptor length.

Return Code 38
Explanation: Thesaurus selection and tree not found.

Return Code 39
Explanation: No search label defined.

Return Code 40
Explanation: Hyper descriptor not found.

Return Code 59
Explanation: Query not found.

Return Code 60
Explanation: Illegal query name.

Return Code 80
Explanation: No proximity search for this chapter.

Return Code 101
Explanation: Operator followed by another operator.

Return Code 102
Explanation: Operand missing before comma.

Return Code 103
Explanation: Comma followed by operator.

Return Code 104
Explanation: Operator followed by right parenthesis.

Return Code 105
Explanation: Operator last token in query.

Return Code 106
Explanation: Operand missing after function.

Return Code 107
Explanation: Function followed by another function.

Return Code 108
Explanation: Function followed by comma.

Return Code 109
Explanation: Function followed by left parenthesis.

Return Code 110
Explanation: Function followed by right parenthesis.

Return Code 111
Explanation: Function last item in query.

Return Code 112
Explanation: Comma followed by function.

Return Code 113
Explanation: Comma followed by comma.

Return Code 114
Explanation: Comma followed by left parenthesis.

Return Code 115
Explanation: Comma followed by right parenthesis.

Return Code 116
Explanation: Comma followed by end of query.

Return Code 119
Explanation: Left parenthesis followed by operator.

Return Code 120
Explanation: Left parenthesis followed by comma.

Return Code 121
Explanation: Left parenthesis followed by right parenthsis.

Return Code 122
Explanation: Left parenthesis followed by end of query.

Return Code 123
Explanation: Right parenthesis followed by comma.

Return Code 126
Explanation: Comma as first token not allowed.

Return Code 127
Explanation: Right parenthesis as first token not allowed.

Return Code 128
Explanation: Empty request.

Return Code 131
Explanation: Reference number followed by comma.

Return Code 135
Explanation: Function followed by reference number.

Return Code 136
Explanation: Comma followed by reference number.

Return Code 140
Explanation: Operator as first token not allowed.

Return Code 141
Explanation: Formatted field followed by operator.

Return Code 142
Explanation: Formatted field followed by function.

Return Code 143
Explanation: Formatted field followed by comma.

Return Code 145
Explanation: Formatted field followed by right parenthesis.

Return Code 146
Explanation: Formatted field followed by end of query.

Return Code 147
Explanation: Formatted field followed by reference number.

Return Code 148
Explanation: Formatted field followed by formatted field.

Return Code 149
Explanation: Relational operator followed by operator.

Return Code 150
Explanation: Relational operator followed by function.

Return Code 151
Explanation: Relational operator followed by comma.

Return Code 152
Explanation: Relational operator followed by left parenthesis.

Return Code 153
Explanation: Relational operator followed by right parenthesis.

Return Code 154
Explanation: Relational operator followed by end of query.

Return Code 155
Explanation: Relational operator followed by formatted field.

Return Code 156
Explanation: Relational operator after relational operator.

Return Code 157
Explanation: Relational operator after operator.

Return Code 158
Explanation: Formatted field after function.

Return Code 159
Explanation: Relational operator after function.

Return Code 160
Explanation: Formatted field after comma.

Return Code 161
Explanation: Relational operator after comma.

Return Code 162
Explanation: Relational operator not after formatted field.

Return Code 163
Explanation: Relational operator after left parenthesis.

Return Code 165
Explanation: Relational operator after right parenthesis.

Return Code 166
Explanation: Relational operator as first token.

Return Code 168
Explanation: Relational operator after reference number.

Return Code 170
Explanation: Reference number after relational operator.

Return Code 171
Explanation: Sort clause not in place.

Return Code 173
Explanation: Formatted field expected after sort.

Return Code 174
Explanation: Right parenthesis without left parenthesis.

Return Code 175
Explanation: Only three sort fields allowed.

Return Code 176
Explanation: Sort field is missing.

Return Code 177
Explanation: Wrong number of words for formatted fields.

Return Code 179
Explanation: Word truncation is not allowed.

Return Code 180
Explanation: Error while scanning source text.

Return Code 181
Explanation: Max. number of words in document exceeds record capacity.

Return Code 182
Explanation: Error while reallocating incore vocabulary buffer.

Return Code 183
Explanation: Internal error: Incore vocabulary buffer not found.

Return Code 184
Explanation: Formatted field as a default chapter.

Return Code 187
Explanation: Two values expected after relational operator ”BETWEEN”.

Return Code 188
Explanation: Left parenthesis without right parenthesis.

Return Code 194
Explanation: Number of query elements exceeds number of pre-calculated elements – rework your query.

Return Code 195
Explanation: Internal buffer overflow. Query too long.

Return Code 196
Explanation: Non numeric items.

Return Code 197
Explanation: Storage allocation failed during syntax analysis.

Return Code 198
Explanation: Storage allocation failed during query execution.

Return Code 199
Explanation: Internal error during syntax analysis.

Return Code 200
Explanation: Invalid position indicator.

Return Code 201
Explanation: Internal error during query execution.

Return Code 202
Explanation: Internal error during load batch.

Return Code 203
Explanation: Internal error during highlighting process.

Return Code 204
Explanation: No ”D3” field in DSFNR.

Return Code 205
Explanation: Invalid ”D3” field length in DSFNR.

Return Code 206
Explanation: Word not found in vocabulary.

Return Code 207
Explanation: Cannot open output file.

Return Code 208
Explanation: Cannot write output record.

Return Code 209
Explanation: Cannot close output file.

Return Code 210
Explanation: Error in DSA call.

Return Code 220
Explanation: Syntax error in ”DUE” call.

Return Code 221
Explanation: Error in user exit call. Illegal output code.

Return Code 222
Explanation: Error in user exit call. ”DSA” area not found.

Return Code 223
Explanation: Error in ”CQR” call. Illegal action code.

Return Code 224
Explanation: No active user exit or user function found.

Return Code 225
Explanation: Main query was released by the user exit.

DYP and BC Return Codes

Return Code 400
Explanation: Unknown keyword within ”DYP” parameter.

Return Code 401
Explanation: Invalid use of ”DYP” parameter.

Return Code 402
Explanation: Unexpected end of ”DYP” parameter.

Return Code 403
Explanation: Invalid document file parameter (DFNR=).

Return Code 404
Explanation: Invalid document summary file parameter (DSFNR=).

Return Code 405
Explanation: Invalid vocabulary file parameter (VFNR=).

Return Code 406
Explanation: No space to allocate buffer for VFNR file FDT.

Return Code 407
Explanation: Cannot reallocate buffer of VFNR file FDT.

Return Code 408
Explanation: Invalid database id. (DBID=).

Return Code 409
Explanation: Invalid database type (DBTYPE=).

Return Code 410
Explanation: Invalid concatenation character (CONCAHR=).

Return Code 411
Explanation: Invalid TRS error prefix number (ERRPRE=).

Return Code 412
Explanation: Invalid limit of vocabulary set (MAXVSET=).
Return Code 413
Explanation: Maximum number of words in doc invalid (MAXWORD=).

Return Code 414
Explanation: Maximum number of aspects for one word invalid (NUMASPCT=).

Return Code 415
Explanation: Invalid length of Adabas password (PASSWORD=).

Return Code 416
Explanation: Invalid length of user information for Adabas control block (TID=).

Return Code 417
Explanation: Invalid set prefix character (SETCHAR=).

Return Code 418
Explanation: Invalid truncation character (TRUNCHAR=).

Return Code 419
Explanation: Start isn for loader invalid (STARTISN=).

Return Code 420
Explanation: Invalid word length (WORDLEN=).

Return Code 421
Explanation: Invalid document id. record key (DOCID=).

Return Code 422
Explanation: No space to allocate buffer for FFE area.

Return Code 423
Explanation: No space to allocate buffer for formatted fields.

Return Code 424
Explanation: Field not in formatted field table.

Return Code 425
Explanation: Invalid text isn's group name (TEXT=).

Return Code 426
Explanation: Invalid document set name (DOCSET=).

Return Code 427
Explanation: Invalid vocabulary set name (WORDSET=).

Return Code 428
Explanation: Invalid user scan id. for the "SCA" call (USCANID=).

Return Code 429
Explanation: Invalid "WAIT ON HOLD" parameter (WH=).

Return Code 430
Explanation: Invalid default file parameter (DEFFILE=).

Return Code 431
Explanation: Max. document id. length exceeded.

Return Code 432
Explanation: Only "WORD" as parameter allowed (INDEX=).

Return Code 433
Explanation: Invalid parameter (INDEX=).

Return Code 434
Explanation: Invalid parameter (INDEX=).

Return Code 435
Explanation: Invalid parameter (INDEX=).

Return Code 436
Explanation: Invalid search label length (SEARCHLB=).

Return Code 437
Explanation: Only 20 search labels allowed (SEARCHLB=).

Return Code 438
Explanation: Default proximity operator invalid (DEFOPER=).

Return Code 439
Explanation: Invalid parameter (FUNCTION=).

Return Code 440
Explanation: Cannot allocate buffer for function definition.

Return Code 441
Explanation: Invalid parameter (FUNCTION=).

Return Code 442
Explanation: Invalid parameter (FUNCTION=).

Return Code 443
Explanation: Not a valid function (FUNCTION=).

Return Code 444
Explanation: Not a valid function (FUNCTION=).

Return Code 445
Explanation: Invalid parameter (FUNCTION=).

Return Code 446
Explanation: Invalid parameter (FUNCTION=).

Return Code 447
Explanation: Invalid parameter (FUNCTION=).

Return Code 448
Explanation: Invalid Adabas error prefix number (ERRADA=).

Return Code 449
Explanation: Invalid incore vocabulary name (INCVOC=).

Return Code 450
Explanation: Invalid parameter for batch loader (LOADER=).

Return Code 451
Explanation: Invalid highlight algorithm parameter (HIGHLIGHT=).

Return Code 452
Explanation: Invalid root file parameter (RFNR=).

Return Code 453
Explanation: Invalid vocabulary prefix char (WORDPREF=).

Return Code 454
Explanation: Invalid User Exit error prefix number (ERRUSE=).

Return Code 455
Explanation: Invalid multi call parameter (MULTICALL=).

Return Code 456
Explanation: Invalid parameter (HOLDWORD=).

Return Code 457
Explanation: Invalid parameter (INVNUM=).

Return Code 458
Explanation: Invalid parameter (FFTRANS=).

Return Code 459
Explanation: Invalid parameter (JFNR=).

Return Code 460
Explanation: Invalid parameter (JREF=).

Return Code 461
Explanation: No DYP during an ADD loop allowed.

Scanner Return Codes

Return Code 300
Explanation: Class name is more than 6 characters long.

Return Code 301
Explanation: More than 16 classes listed.

Return Code 302
Explanation: Mode must be ”A”, ”F”, ”P”, ”Q” or ”T”.

Return Code 303
Explanation: Class not found in list defined.

Return Code 304
Explanation: Keyword ”CLASS=” missing.

Return Code 305
Explanation: No classes have been defined. No ”SCTS” call done.

Return Code 306
Explanation: Problem allocating ”DSA” space.

Return Code 307
Explanation: Bad translation code specified.

Return Code 308
Explanation: Undefined lable referred to.

Return Code 309
Explanation: Undefined action.

Return Code 310
Explanation: Undefined type.

Return Code 311
Explanation: Undefined mode.

Return Code 312
Explanation: Area size too small for save.

Return Code 313
Explanation: Bad action specified for ”SCTK” call.

Return Code 314
Explanation: Substrings too long to be saved.

Return Code 315
Explanation: Token too long.

Return Code 316
Explanation: Keyword ”CHAR=” missing.

Return Code 317
Explanation: Comma must appear.

Return Code 318
Explanation: Keyword ”LC=” missing.

Return Code 319
Explanation: Keyword ”UC=” missing.

Return Code 320
Explanation: Wrong class identifier.

Return Code 321
Explanation: Left parenthesis must appear.

Return Code 322
Explanation: Right parenthesis must appear.

Return Code 323
Explanation: Unknown class.

Return Code 324
Explanation: Incorrect keyword.

Return Code 325
Explanation: More than 99 keywords listed.

Thesaurus Return Codes

Return Code 251
Explanation: Son/father combination exists already in the set id. or thesaurus.

Return Code 252
Explanation: The son/father combination would create a loop.

Return Code 253
Explanation: The son/father combination does not exist.

Return Code 254
Explanation: No father found for this term.

Return Code 255
Explanation: The set id. does not exist in the thesaurus.

Return Code 256
Explanation: The term does not exist in the set id. or thesaurus.

Return Code 257
Explanation: No ”START” was done for this listing.

Return Code 258
Explanation: Trying to add different type to a set id.

Return Code 259
Explanation: Set id. is not a synonym type.

Return Code 260
Explanation: Son or set id. cannot be an empty record.

Return Code 261
Explanation: Set id. is not a tree type.

Return Code 263
Explanation: End of list.

INDEX