System Librarian's Guide - Indexing

Version 22

ExLibris The bridge to knowledge

CONFIDENTIAL INFORMATION

DISCLAIMER

TRADEMARKS

Document released: March 2016

Web address: http://www.exlibrisgroup.com

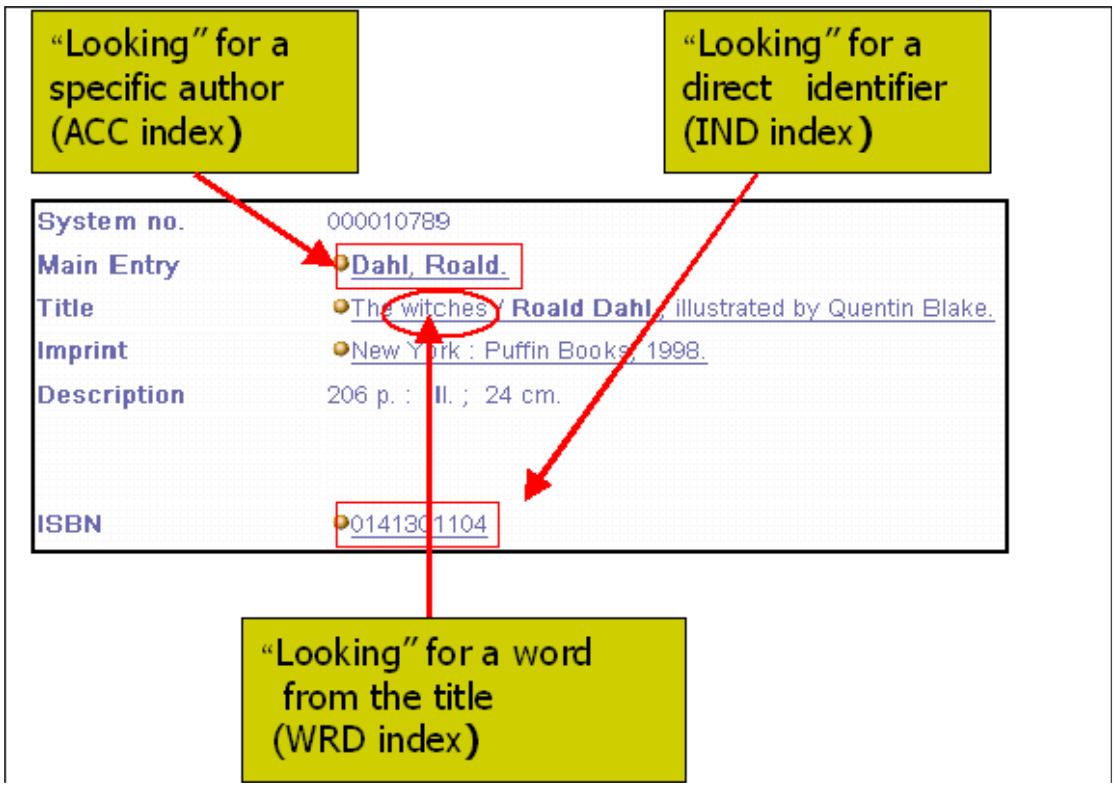
Table of Contents

1 Indexing Overview

ALEPH allows various definitions of access paths (indexes) to document records. These definitions can be tailored to suit the needs of each application.

Indexes can be based on specific fields, subfields, on a combination of fields, or on individual words from specific document fields.

There are three major ways of accessing the bibliographic database, via three types of indexes: Headings Index, Word Index and Direct Index. Each of these types of indexes has specific subindexes attached. For example, among the subindexes attached to the Headings (ACC) index are the TIT and AUT subindexes.



Indexes are used by end users for OPAC searches and by librarians for internal needs, such as catalogers' lists and record retrieval to produce various reports.

This chapter is intended mainly for system librarians. It provides an overview of data access methods. It includes index definitions and types, index specifications, indexing processes, expand programs.

Note that the various indexing services can be accessed via the Cataloging GUI by opening the Services menu and selecting an option from the Build Indexes to Catalog submenu.

2 Indexing Process

2.1 Defining Indexes in ALEPH

There are three basic steps for defining indexes in ALEPH:

- Defining codes and names of the indexes in `tab00.lng`.
- Assigning field tags to the various index codes to define the connection between the record fields and the indexes.
- Placing the index in `tab/pc_tab_sear.lng` and in `alephe/www_f_lng/` files to enable searching of the index via the Search function in the GUIs and Web OPAC.

2.1.1 Defining a New Index

To define a new index:

1. Decide which type of index you are creating: Direct, Word or Headings.
2. Add the Index to `xxx01/tab/tab00.lng`. Look at `tab00.lng` and decide on a code for the new index (make sure the code is not already in use).

Notes

- Although the code can be up to five characters, by convention it is usually only three characters.
- If the index is a Direct index, specify "IND" for the index type.
- If the index is a Headings index, specify "ACC" for the index type.
- If the index is a Words index, find the next "W-nnn" value and use that for the index type. The -nnn numbers must be unique and consecutive.
- The Words index type should always start with the letter W.
- **The system always provides SCAN and FIND access by system number and FIND access by barcode. Therefore, although they do not need to be defined in the indexing table (tab11), they must be defined here in order to define the index name in column 11.**

2.1.2 Assigning Fields to an Index

To assign fields to an index:

1. Add the Index to `tab11`.
2. Include the code you specified in `tab00.lng` in the entry for each field in `xxx01/tab/tab11` which you want to have sent to this index.
3. If you specified "IND" in `tab00.lng`, put the entry in `tab11_ind`.
4. If you specified "ACC", put the entry in `tab11_acc`.
5. If you specified "W-nnn", put the entry in `tab11_word`.

2.1.3 Adding an Index to pc_tab_sear.lng

If you want an index to be searchable through the Search functionality in the GUIs, then add it to `xxx01/tab/pc_tab_sear.lng`.

If you want to access the index in a browse mode, include it in the "SC" (Scan) section. The index terms are displayed in alphabetical order, and you can scroll forward. Headings indexes are particularly well-suited for this mode.

If you want to be able to retrieve a set of records that match the index terms, include it in the "FI" (Find) section. WORD, SYS and BAR indexes are particularly well-suited for this mode.

2.1.4 Add an Index to Web Include files

If you want the index to be searchable in the Web OPAC in order to display an alphabetically-arranged scan list, add the index to the `scan-include-2` file in the `alephe/www_f_lng/` directory.

In order to perform the retrieval of records which are presented in a set, add the index either to the `scan-include-2` file or to the `find-code-include` file in the `alephe/www_f_lng/` directory.

2.2 Ongoing Indexing

The UE_01 background process updates the Headings Index, Word Index and the Logical Bases Counter (Z0102); the Direct Index is updated immediately.

When a cataloging record is added or updated, its system number is placed in the Z07 table. UE_01 checks the Z07 table to detect new/updated records waiting for indexing.

Once the procedure has updated the heading indexes, the Z07 record is deleted and a Z07a record is created. Once the logical bases counter is updated and the word index is created, the Z07a record is deleted. A smoothly running system should not have many records in the Z07 and Z07a tables.

The ue_01 process includes three separate sub-processes:

- ue_01_a: This process is responsible for the update of the headings (in addition it updates other and indexes and tables such as the Z13, Z00R, Z101, and so on).
- ue_01_word_parallel: This process is responsible for the word index (for performance considerations). It builds the words in parallel to other indexes.
- ue_01_z0102_index: This process is responsible for the Z0102 records (logical bases counter).

The logical bases counter (Z0102 records) was updated by the ue_08 process. Note that the performance of the ue_08 process has been greatly improved by the introduction of the ue_01_z0102_index sub-process.

The UE_01 process can be initiated in two ways:

- Invoking UTIL E/1 when the system is started up, or
- Setting the process for automatic startup in the `aleph_startup` file of the `$alephe_root` directory. If the UE_01 daemon is not running, you cannot browse or search for recently added records. However, you can look them up using their system number, which is indexed automatically in the IND index.

For big libraries experiencing backload in ongoing indexing, an improvement has been added to ue_01. The word indexing part is split into 10 processes (thus ue_01 includes 13 separate sub-processes).

In order to implement this functionality, execute the following steps:

1. Stop ue_01: `util e/2`.
2. Build 10 new temporary tables used internally by the processes:
   a. Add in `usm01/file_list` the following line:
      ```
      TAB z98t                2K              0K              TS1D
      ```
   b. Run `util a/17/1` with table name: z98t (it builds all 10 tables).
3. Build PL/SQL procedures. For instance, to run `ue_01` in USM01, run the following:
   ```
   cd $alephm_root/sql_tab/
   s+ usm01
   SQL> @z98t_proc
   SQL> exit
   ```
4. Set the flag activating the mechanism in [`bib_library`]`/prof_library`:
   ```
   setenv word_queue Y
   ```

Note:

For all other libraries (AUT, ADM, ILL and HOL libraries) that do not use this mechanism, deactivate it by setting the "word_queue" flag to "N" in `prof_library`:

```
setenv word_queue N
```

5. Execute the following command: `dlib USM01`.
6. Restart ue_01: `util e/1`.

2.2.1 Prioritizing the Ongoing Indexing

The ongoing indexing that the UE_01 background process manages, uses a set of priority rules so that some type of record updates may be indexed before others. Some of these priority rules are hardcoded, and some may be configured.

As a rule, every change in the catalog triggers a Z07 record creation. The Z07 record is the trigger for the UE_01 background process management. The lower the sequence that Z07 is assigned, the higher the record's priority is for UE_01 management. The following rules apply to setting the Z07 sequence:

- Newly cataloged records are assigned the sequence 1990, regardless of whether they are created online or offline. All new records are indexed on a first created-first indexed basis. This is a hardcoded mechanism.
- Updates in existing catalog records, regardless of whether the records are updated online or by the OCLC server, get assigned the sequence 1998. This will make new records receive a higher priority over changes in existing records (as 1990 is less than 1998). This is a hardcoded mechanism.
- When running the 'Load ALEPH Sequential MARC Records (manage-18)' batch job, the priority of the cataloged records may be set using the job's 'Override Indexing Priority' parameter. For example, if the parameter is set to 1985 then the records loaded by the batch will receive a higher priority than newly online cataloged records (as 1985 is less than 1990).
- All loading batch jobs may be assigned a priority by setting an environment variable in the $alephe_root/aleph_start.private, which sets the indexing priority for records that are created by the batch. For example:

  setenv z07_p_manage_40 2000

  setenv z07_p_manage_18 2001

  setenv z07_p_manage_180 2001

  setenv z07_p_file_90 2002

  setenv z07_p_file_93 2002

  setenv z07_p_file_95 2003

  setenv z07_p_file_96 2004

  setenv z07_p_file_97 2005

  setenv z07_p_file_98 2005

  setenv z07_p_file_99 2005

- In addition, the OCLC server and the UE_08 daemon may be set with priorities, for example:

  setenv z07_ue_03 2006

  setenv z07_oclc_server 2007

  setenv z07_ue_08 2008

Notes:
The priority that is set in the 'Override Indexing Priority' parameter of the manage-18 batch jobs overrides a value that is set in the z07_p_manage_18 variable.

The priority that is set in these environment variables overrides the hardcoded definitions listed in the previous points.

2.3 Building Indexes

Indexes are built by three batch procedures:

- p_manage_02 (Headings index)

- p_manage_01 (Word index)
- p_manage_05 (Direct index)

The batch-building procedures must be run either after conversion and/or after modifying the indexing tables (for example, `tab11`, `tab00.lng`, and so on) to update the index records.

3 Headings Index

This chapter includes the following sections:

Filing of Headings

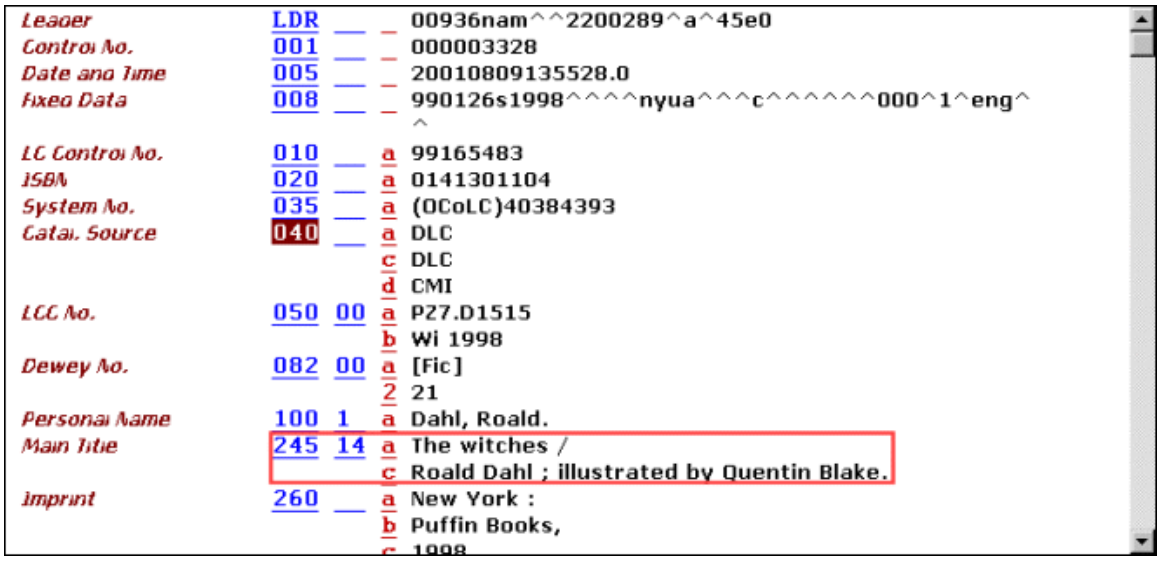
Main Headings Services

Other Headings Services

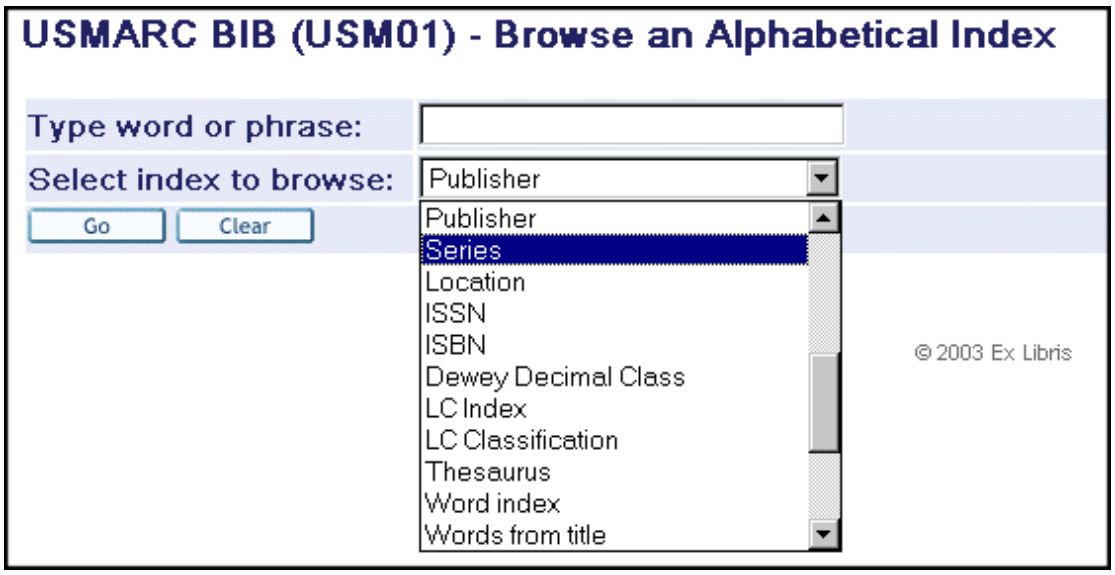
Linking Process

The Headings Index creates a list of entries that can be browsed by the patron or by a librarian in the Web OPAC. The Headings Index is also sometimes called the Browse List or ACC List.

Headings indexes are phrases from a record field such as author, title, subject, publishers, and so on. A Headings Index term can be either an entire field or one or more specific subfields.



Each library decides which fields of the bibliographic record form the basis for the Browse function in the Web OPAC. For example, you can decide to provide the ability to browse by authors, titles, publishers, and so on.



The Headings or Browse index can be accessed through the Browse function in the Web OPAC or via the Search functionality in the GUIs.

Each field or subfield is assigned to a specific headings group. For example, all types of titles can be assigned to the title headings group. Subjects can be assigned to a different "sub-index" for subjects.

3.1 Filing of Headings

Headings are filed (organized) in the browse list according to the "filing text" of the heading.

The filing text is built in three steps, based on the field text:

Display text (usually data taken from the bibliographic record without the final punctuation).

Normalization of the display text.

Normalized text to filing text.

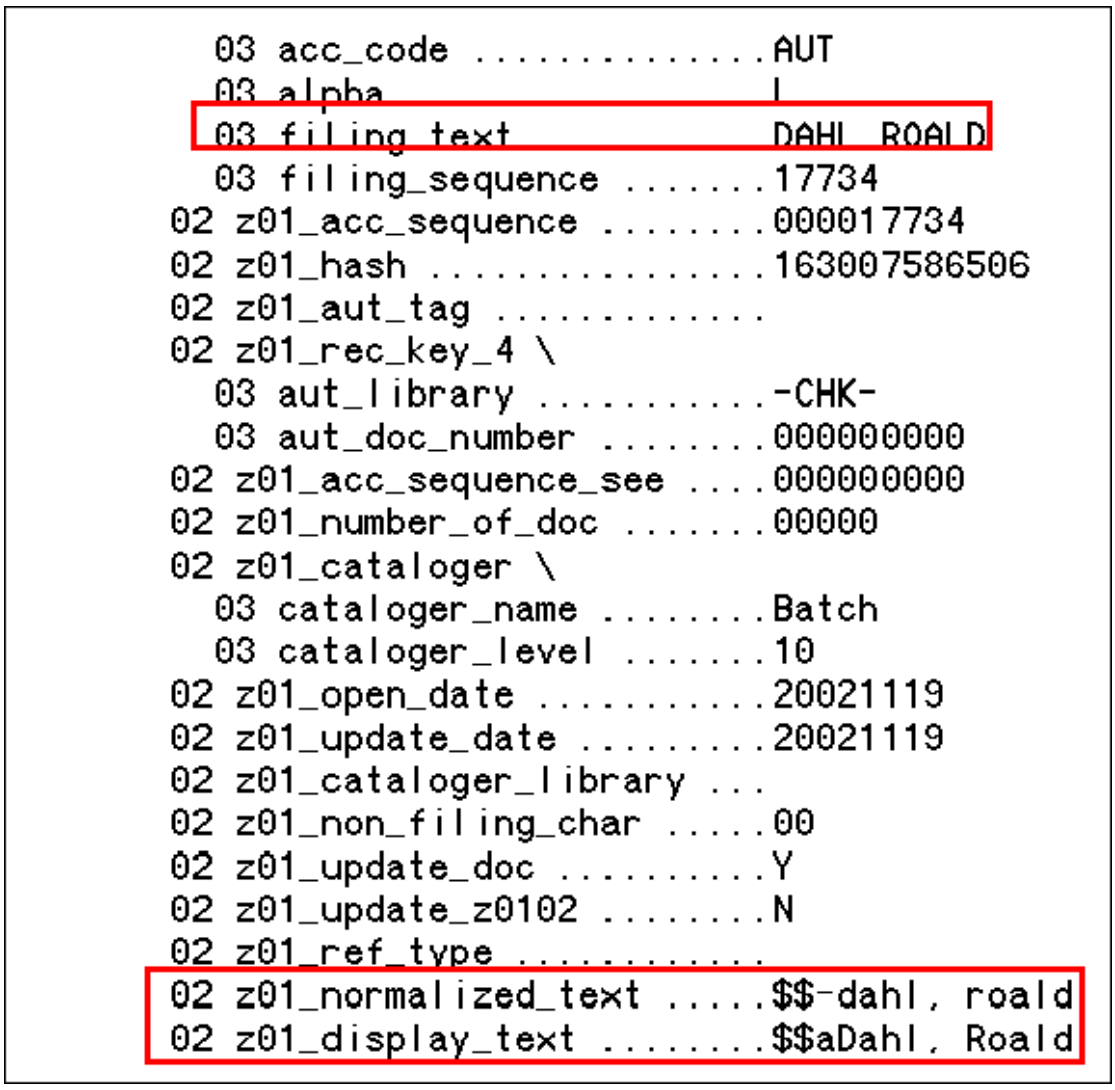
Conversion takes place according to the procedures defined in the tab_filing table of the library's tab directory and in the tab_character_conversion_line table of the `alephe/unicode` directory.

```
z01_display_text .........$$aDahl, Roald

z01_normalized_text .....$$-dahl, roald

z01 filing text ...........DAHL ROALD
```

3.1.1 Normalization

Normalization refers to the process whereby diacritics, most punctuation marks, special characters, and case differences are stripped from access fields (headings) for the purpose of determining the uniqueness of headings.

The normalized form of the headings is built according to the rules defined in the library's `tab_filing` table:



To view the whole list of filing routines and an explanation of normalization routines refer to Sorting and Word Breaking on page 36.

The aims of normalization are:

To treat the same headings alike.

To ensure that each unique heading is stored only once in the headings index.

To distinguish headings that are different by means of unique identifiers.

3.1.2 Database Tables Involved

Doc

The document table that is being indexed (in the case of a bibliographic library, this is the bibliographic record).

Z01

The Z01 table is a list of headings (entries) derived from information in the bibliographic record through which the user can browse in the Web OPAC and via the Search functionality in GUIs. The Headings Index is also sometimes called the Browse List or the ACC List. Users can browse records by Author, Title, Subject or any other category defined by the library.

Z01 contains the headings according to the indexing defined in the tab00.lng and tab11_acc tables of the library's tab directory.

Z02

Z02 holds links between the Z01 - Access Headings - and the bibliographic records.

Z07

When a cataloging record (bibliographic, authorities, holdings, and so on) is created or updated, its system number is placed in the Z07 table. The Z07 is used by the system for controlling the updating of index files. Indexes are updated by the background job, initiated by UTIL E/1 (Update Doc Indexes - UE_01).

3.2 Main Headings Services

The Headings index is created by p-manage-02, enriched by UE_08, and updated by UE_01.

3.2.1 Update Headings Index (manage-02)

This service updates the Headings index of the database.

3.2.2 Pre-Enrich Bibliographic Headings Based on the Authority Database (manage-102)

p_manage_102 copies the headings from authority records into the bibliographic file Z01. This makes the long, complete run of UE_08, which matches headings with authority records, unnecessary.

For further information about this process, refer to the Authorities - Batch Jobs for Authority Enrichment and Correction of Bibliographic Libraries section of the Authorities chapter in the ALEPH System Librarian’s Guide.

3.2.3 Subject Subdivisions

For information about subject subdivisions, refer to the Authorities - Control section of the Authorities chapter in the ALEPH User Guide.

3.2.4 When to Run the Headings Index

The "Update - Headings Index" service must be run after:

A new code has been added to an already existing index,

A change has been made to the `tab00.lng` table or `tab11_acc` table that affects the Headings Index.

A change has been made in `tab_filing` that affects the display or normalized form of the heading.

The "rebuild" option in the "Procedure to Run" field of the "Update Headings Index" service must be run after making changes that affect already existing index entries.

We recommend rebuilding the Headings Index periodically, using the "Update Headings Index" service.

Note

After you run this service using the "Rebuild" option, always run the Alphabetize Long Headings (manage-17) service.

3.3 Other Headings Services

3.3.1 Alphabetize Headings (p_manage_16)

This service alphabetizes the headings according to the rules for alphabetization that are stored in the `tab00.lng` table and in the `tab_filing` table.

These rules create a "filing text" by which the heading is alphabetized. The headings are then alphabetized according to the first 69 characters of the filing text of each entry.

When to Run this Service

This service must be run any time the rules for alphabetization have been changed in the `tab00.lng` table or in the tab filing table (including character conversion tables). For example, you may decide that you now want to alphabetize under "ue" instead of "u". If the change in rules affects display or normalized texts, p-manage-02 must be run instead.

Note

After you run this service, always run the Alphabetize Long Headings (manage-17) service.

3.3.2 Alphabetize Long Headings (p_manage-17)

This service alphabetizes those headings whose filing texts are longer than 69 characters.

When to Run this Service

Run this service after conversions or if you suspect that there is a problem with the filing of long headings. In addition, if at any time, the rules for alphabetization are changed in the `tab00.lng` table or in the `tab_filing` table (including character conversion definitions), all the headings should be re-alphabetized to reflect the new rules. First run the Alphabetize Headings - Setup (manage-16) service, then run the Alphabetize Long Headings service.

3.3.3 Build Counters for Logical Bases (manage-32)

This service builds the counters for logical bases.

Counters for logical bases (Z0102) can be used to make browsing from the Web OPAC more efficient when scanning (browsing) logical bases which are less than 10% of the total database. "Y" in col. 8 of `./alephe/tab_base.lng` determines that the logical base must have Z0102 records built. If the library does not use logical bases, or if `tab_base.lng` does not include "Y" in col. 8, this section is irrelevant.

When a logical base is being browsed, the system uses these counters to determine whether or not to display the heading without having to retrieve the documents attached to the heading, read them and then determine how to proceed.

When to Run this Service

This service must always be run after building the headings index (manage-02). The database tables involved are:

Z01

Z02

Z0102

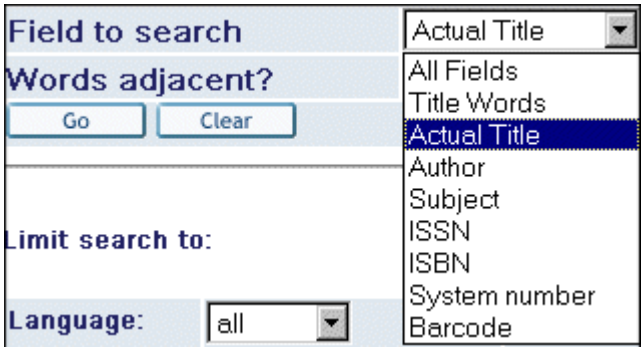
Z0102

Entries in the Z0102 table are built for each heading and for each logical base. The table includes the filing text, the acc-sequence identifier of the heading and a counter of the relevant documents attached to the heading. The pointer to the documents is still stored in the Z02 (ACCDOC) table.

3.3.4 Update Brief Records (manage-35)

This service updates and creates brief records. The Brief record display format is used to sub-arrange the records that are attached to a particular heading. These records are built according to the record's format, the headings index and the field of origin.

The structure of the brief record is defined in the `tab_z0101` and the `tab_z0101_text` tables of the library's `tab` directory.



The database tables involved are:

Z0101

Note that libraries that work with the Z0101 format must add the last-z0101-sequence counter to UTIL G/2.

3.3.5 Update Short Bibliographic Records (manage-07)

This service updates the Short Bibliographic Records of the database. The purpose of the Short Bibliographic Record is to provide bibliographic information in an efficient and timely manner, particularly for instances where bibliographic information is an adjunct to administrative information.

The Short Bibliographic Record is built by the system, according to the setup of the tab22 table (in the library's tab directory), when records are uploaded into the database (when the indexing parameter is set to 'Full'), or when records are added or updated through the Cataloging module.

A Short Bibliographic Record is an abbreviated version of the bibliographic record in standard Oracle table format. It can contain up to seven fixed (system-defined) fields (year, call number, call number key, author, title, imprint and ISBN/ISSN) each limited to 100 characters (except for the call number key field that is up to 80 characters). Additionally, it contains up to fifteen user-defined fields, each limited to 500 characters.

When to Run this Service

Run this service after making a change in the tab22 table that affects the Short Bibliographic Records.

The database tables involved are:

Z13

3.3.6 Create Sort Keys (manage-27)

Sort keys are used in the Web OPAC for sorting a set of records. The keys are data extracted from a field in the bibliographic record (for example, title, year). They are built in accordance with the rules set in `tab_filing` and `tab_character_conversion_line`.

3.3.7 Delete Unlinked Records (manage-15)

This service deletes headings that are not linked to records. Each modification of cataloging records creates new headings in the system. The old heading records are kept but the link to the record is deleted. This function deletes all such extra, unlinked headings.

Note that this process does not delete heading records which have an authority link. This is in order to keep the cross-references which are not linked to the document directly.

3.4 Linking Process

The process that triggers the building of the bibliographic heading - authority record connection is the UE_08 daemon. This process is initiated by UTIL E/8.

The UE_08 procedure checks new headings in the bibliographic library against the authority library and adds cross-references and/or multi-lingual equivalents to the bibliographic headings table.

A heading becomes "authorized" when a direct match is made between a heading [Z01] from the bibliographic database and a record from the authority database. The authority record is found through the authority library's headings index.

For further information about this process, refer to The Authority Database as Search Aid section of the Authorities chapter in the ALEPH User Guide.

4 Word Index

This chapter includes the following sections:

Defining Words

Database Tables Involved

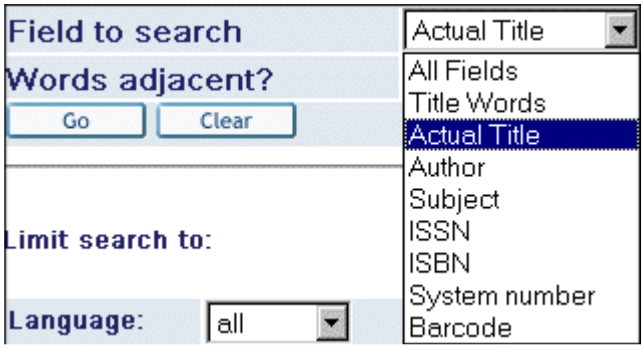
When to Update the Word Index

The Word Index contains a list of words that appear in specific fields of the bibliographic records in the database.

When a patron or librarian uses the Search function on a Word index in the Web OPAC, the system retrieves all documents containing the keyword(s) entered by the user.

Words are assigned to specific word groups. Thus, all words from the various types of title can be assigned to the "words from titles" group. Words from subjects can be assigned to a different word group.

Web OPAC searches can apply to the general word index or to any specified sub-index:

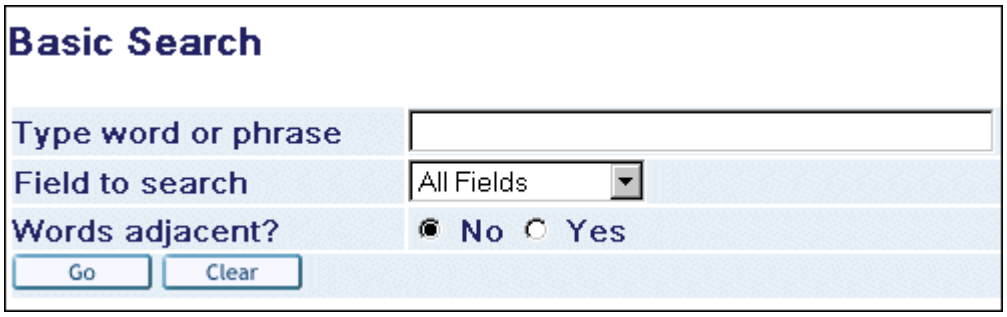


The system extracts each unique word from the specific fields of the record, stores it in the Word file, and maintains pointers to the document:

The extracted words are stored in the Z97 table that contains the word dictionary. The word dictionary is a list of all the searchable words derived from information in the document record.

When the user performs a Find/Search request from the Web OPAC, the system checks the Word Index to retrieve all documents containing the keyword(s) entered by the user.

The Word Index is usually used for the Search function in the Web OPAC and in the GUIs:



4.1 Defining Words

The default definitions of a word are:

A character string from blank to blank, or

From the beginning of a line to the first blank, or

From the last blank to the end of a line.

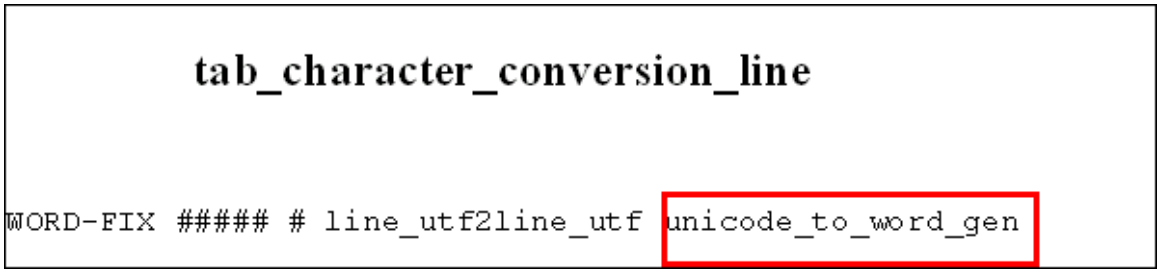
Word Breaking

Word-breaking routines are used to define what will be considered a "word" for the system in special cases (for example, I.B.M). Word-breaking routines are listed in Sorting and Word Breaking on page 36.

Word-breaking routines are specified for each word index group through column 6 of the `tab11_word` table of the library's tab directory.

Character Conversion

In addition to the word-building procedures, after text has been broken into words, a character conversion table is used to define equivalencies for characters. The system uses the character conversion table that is listed in column 5 of the `tab_character_conversion_line` table of the `$alephe_unicode` directory under the WORD-FIX entry:



4.2 Database Tables Involved

Z95

The records of this table contain the list of words in a document. The UE-01 online indexing process writes the Z95 records.

Z97

The Z97 table contains the word dictionary. It contains a list of all the words derived from the document record.

Each library can decide which fields of the document record form the basis for the Find/Search function in the Web OPAC. For example, you might decide to provide the ability to search by words from titles. The system extracts each unique word from the Title fields of the records and creates a unique entry for the word in the Z97 table. The Z97 table contains all the searchable words according to the indexing defined in the library's `tab/tab00.lng` and `tab11_word` tables.

Adjacency Searching

If the environment variable, "setenv ADJACENCY 2" is specified in ../alephe/aleph_start, then word indexing automatically builds word pairs in the Z97-Word Dictionary, for adjacency searching.

Note that the creation of paired words for adjacency searching requires a lot of disk space (four times more per record). On the other hand, it improves the performance of adjacency searching.

Z98

This table contains word document relations. Z98 contains a bitmap-compressed map of word occurrences in documents. The bitmap maintains pointers from the words registered in Z97 to the documents. The WORD3 utility (UTIL/F/4/word3) assists in reading the bitmap. This utility reads the bitmap in order to find the documents that contain word X stored in index Y.

4.3 When to Rebuild the Word Index

The Rebuild Word Index (manage-01) Target service must be run after making changes in the tab00.lng table or in the tab11_word table of the library's tab directory that affects word indexing.

5 Direct Index

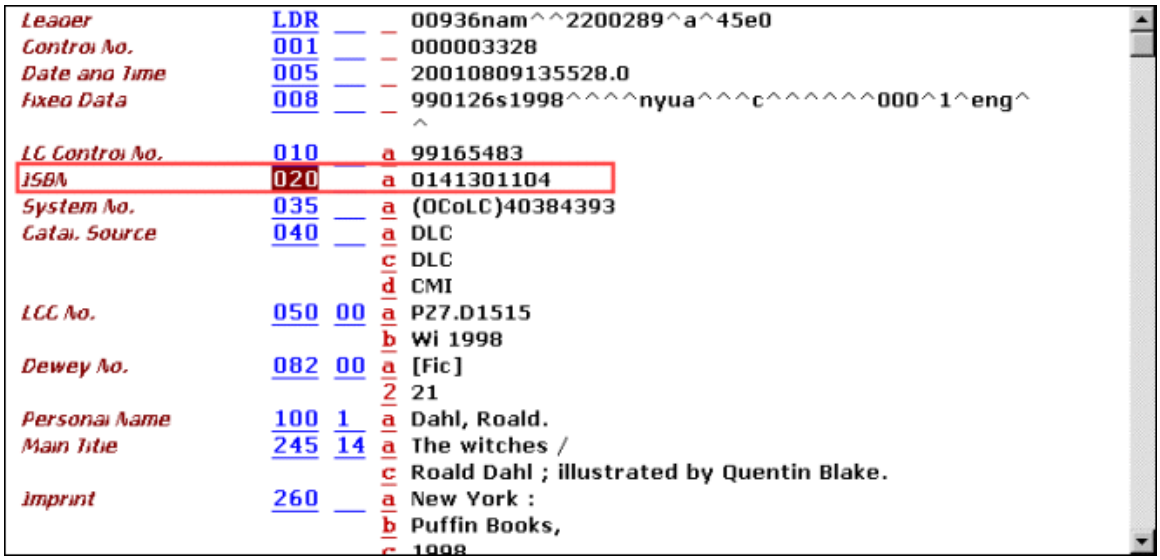
This chapter includes the following sections:

Filing Direct Indexes

When to Update the Direct Index

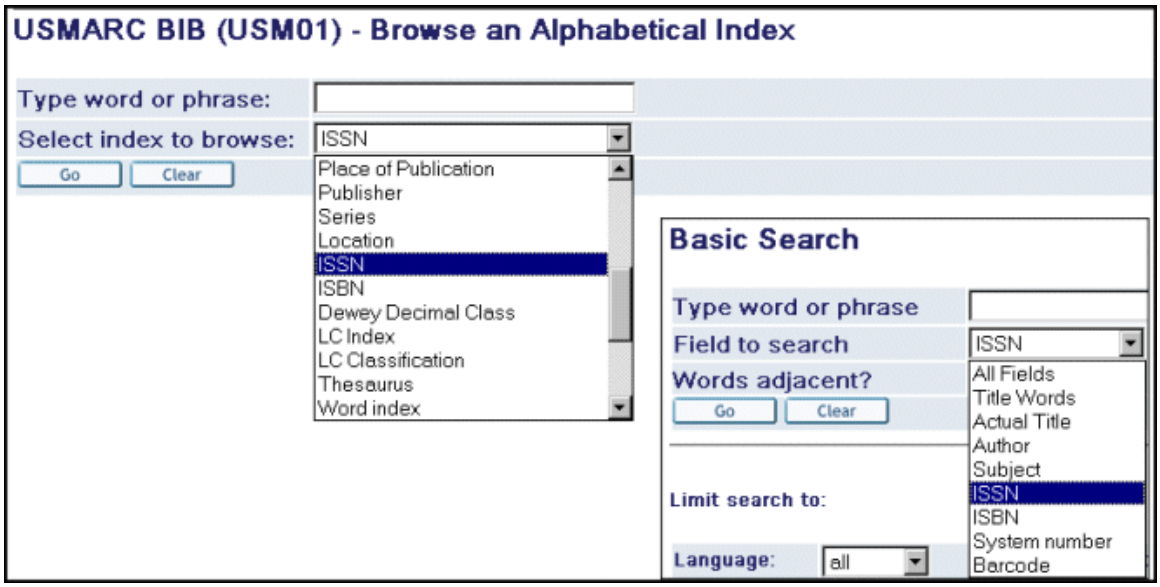
Database Tables Involved

Direct indexes enable the user to retrieve a specific record. A direct index is suited to unique or almost unique identifiers (such as the ISBN, the ISSN and the shelving location) of the record and provides quick access to a record:



Each library can decide which fields of the record are suited for this type of index and search capability.

Direct access functions are available from the menus of the Browse and Search options in the Web OPAC. Browsing a direct index displays the index term and the record title. When, for example, a patron or librarian uses the Browse function in the Web OPAC to locate the ISBN field of the bibliographic record, the system checks the Direct Index to retrieve the exact record containing the search string, or the next closest record. When the Search function is used, the system retrieves the matching record(s) in a set.



5.1 Filing Direct Indexes

Direct indexes are filed according to the filing text of the direct index, which is built according to the rules defined in the `tab_filing` table of the library's `tab` directory.

Filing routines are specified for each direct index through column 5 of the `tab00.lng` table of the library's `tab` directory.

5.2 When to Update the Direct Index

The Update Direct Index (manage-05) service must be run after making a change in the `tab00.lng` table or `tab11_ind` table of the library's tab directory that affects the direct index.

Rebuild
If changes that affect already existing index entries have been made, the Rebuild option in the procedure to run the drop-down list box must be run.

Update
If a new code has been added to an already existing index, or if a new index has been added, use the Update option in the procedure to run the drop-down list box.

Note

The Direct Index must be updated if a large number of records has been uploaded into the database in the "partial" mode and if the indexing has not been performed automatically for those records.

5.3 Database Tables Involved

Z11

The Z11 table contains the direct indexes defined in the `tab11` and in the `tab00.lng` tables of the library's `tab` directory.

Note

Z11 indexing is independent of the ue_01 process; it happens automatically when a bibliographic record is added or updated.

6 APAC Indexing

6.1 Headings Indexing

There are special features for filing Chinese, Japanese, and Korean headings.

Four filing routines are available in order to define whether CJK headings are sorted by Pinyin or by strokes. If these routines are not used, the headings are sorted by Unicode value and no special indexing setup is required. The following are the available CJK filing routines:

- **cjk_pinyin**: This routine adds an exclamation point (!) before each CJK character, translates the characters to pinyin using the Z114 table, and adds the Unicode value in decimal notation. The exclamation point (!) causes the pinyin filing-text to be sequenced separately from regular Latin characters.
- **cjk_stroke**: This routine is similar to the cjk_pinyin routine, except that each character is translated to the stroke value, using the Z114 table.
- **chi_pinyin**: This routine translates each character to pinyin using the Z114 table and adds the Unicode value (in decimal notation) for each character. The Unicode value is added in order to differentiate between different characters that have the same pinyin value. Because the pinyin filing text is sequenced together with regular Latin characters, this routine should be used for browse lists that use the language code from 008 to create separate browse lists (for example, AUTC).
- **chi_stroke**: This routine is similar to the chi_pinyin routine, except that each character is translated to the stroke value, using the Z114 table.

The Aleph default setup is adjusted to sort by Pinyin. Therefore, when multiple CJK languages are used, it is required to create a separate CJK index for each CJK language that is indexed in the catalog and browsed by the user. The following is a description of how this setup is configured.

o tab_expand

```
!   1                   2                              3
!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!>
```

```
ACC       expand_doc_bib_lng_cjk
```

These lines set the headings and words indexing process to run the `expand_doc_bib_lng_cjk` routine. The routine adds the $$9 subfield with the language code from 008/34-36 to each field which contains Chinese, Japanese, or Korean characters. $$9 is then used in `tab11_acc` and `tab11_word` to put the CJK entries into different indexes.

o tab11_acc

```
! 1    2    3     4        5            6           7 8
!!!!!-!!!!!-!-!!!!!!!!!!-!!!!!-!!!!!!!!!!!!!!!!!!!!-!-!
245##       9 chi        TITC  -e468
245##       9 jpn        TITJ  -e468
245##       9 kor        TITK  -e468
245##       9 -          TIT   -e468
```

In this example, each CJK language is set in a separate heading.

o tab00.lng

```
!   2     3   4 5  6  7 8   9   10            11
!-!!!!!-!!!!!-!-!!-!!-!-!---!!-!!!!!-!!!!!!!!!!!!!!!!!!!
H TITC  ACC     51 00       00       Chinese Titles
H TITJ  ACC     52 00       00       Japanese Titles
H TITK  ACC     53 00       00       Korean Titles
H TIT   ACC     11 00       00       Titles
```

In this example, each CJK heading is assigned a separate filing routine.

- **tab_filing**

```
!1 2        3              4
!!-!-!!!!!!!!!!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!
51 F chi_pinyin
52 F jpn
53 F kor
```

This example is coordinated with `tab00.lng` which has "51" as the filing procedure for TITC (Chinese Tiles) and so on.

When creating the filing key, "chi_pinyin" uses Z114 to translate characters to Pinyin and adds the Unicode value of the character after each Pinyin character, in order to differentiate between multiple Chinese characters that share the same Pinyin value.

There is no parallel dictionary available for Japanese and Korean. When creating the filing key for these languages using "jpn" and "kor", the system uses the Unicode value of the characters.

Note

If `tab100` is set to sort headings by Pinyin, but `tab_expand` is not set to add $$9 with the language code, then headings with Chinese characters will interfile with headings in Latin characters, because the headings will be written on the general AUT/TIT lists. However, in this case, filing procedure "cjk _pinyin " can be used to separate the Pinyin words from the regular sequence. It adds an exclamation point (!) before each CJK character, translates the characters to Pinyin using the Z114 table, and adds the Unicode value in decimal notation. This causes the Pinyin filing-text to be sequenced separately from regular Latin characters within the general browse lists.

6.2 Words Indexing

Searching of APAC words is based on

- Normalizing the indexed text
- Defining Segmentation Routines for Indexing
- Defining Segmentation Routines for Searching

6.2.1 Normalizing the Indexed Text

Several normalizing routines may be required prior to indexing the APAC text. For example:

Punctuation Removal
Replace all CJK and Latin punctuation by space. This may be done using the existing "to_blank" routine.

Characters Transliteration
Replace characters by equivalent characters using a transliteration table. This may be done using the existing "char_conv" routine.

Normalizing the indexed text is done by setting tab11_word and tab_word_breaking, for example:

- tab11_word of the BIB library:

```
!1    2     3 4          5                    6  7 8 9     10
!!!!!-!!!!!-!-!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!-!!-!-!-!!!!!-!!!!!!!!!!!!!!
245##                    w                    01     WTI
```

  The WTI Index will contain words from 245 $$w using "01" word-breaking routines.

- tab_word_breaking of the BIB library:

```
!1 2        3                                    4
!!-!-!!!!!!!!!!!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
01 # to_blank             !@#$%^()_={}[]:";<>,.?|\
01 # char_conv            CJK_TO_NORMAL
```

  The "01" word breaking routine removes punctuation using the to_blank routine, and transliterates the characters based on the char_conv routine.

The tab_character_conversion_line table in $alephe_root/unicode must define the char_conv table to be used. For example:

```
!!!!!!!!!!!!!!!!!!!-!!!!!-!-!!!!!!!!-!!!!!!!!!!!!!!!!!!!-!!!!!!!!!!!!!!!!
CJK_TO_NORMAL       ##### # line_utf2line_utf             cjk_to_normal
```

The `cjk_to_normal` table must be defined in the $alephe_root/unicode directory.

6.2.2 Defining Segmentation Routines for Indexing

The tab_word_breaking routine that will be used in the indexing process is defined in column 6 of the relevant tab11_word line. For example:

```
1    2     3 4          5                    6  7 8 9     10
!!!!!-!!!!!-!-!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!-!!-!-!-!!!!!-
!!!!!!!!!!!!!
245##                    w                     01     WTI
```

The tab_word_breaking routine will, in addition to the above described normalization routines, also define the required segmentation routine. For example:

```
!1 2        3                                    4
!!-!-!!!!!!!!!!!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
01 # to_blank              !@#$%^()_={}[]:";<>,.?|\
01 # char_conv             JAPANESE_TO_NORMALIZED
01 # morpheme_index
```

See the Suggested APAC Indexing and Searching Segmentation Routines Setup on page 29 for examples of segmentation routines.

6.2.3 Defining Segmentation Routines for Searching

The tab_word_breaking table is used to define the policy for segmentation of a given string during the searching process.

The following tab_word_breaking sections are reserved by the system for processing the search string when a FIND action is invoked:

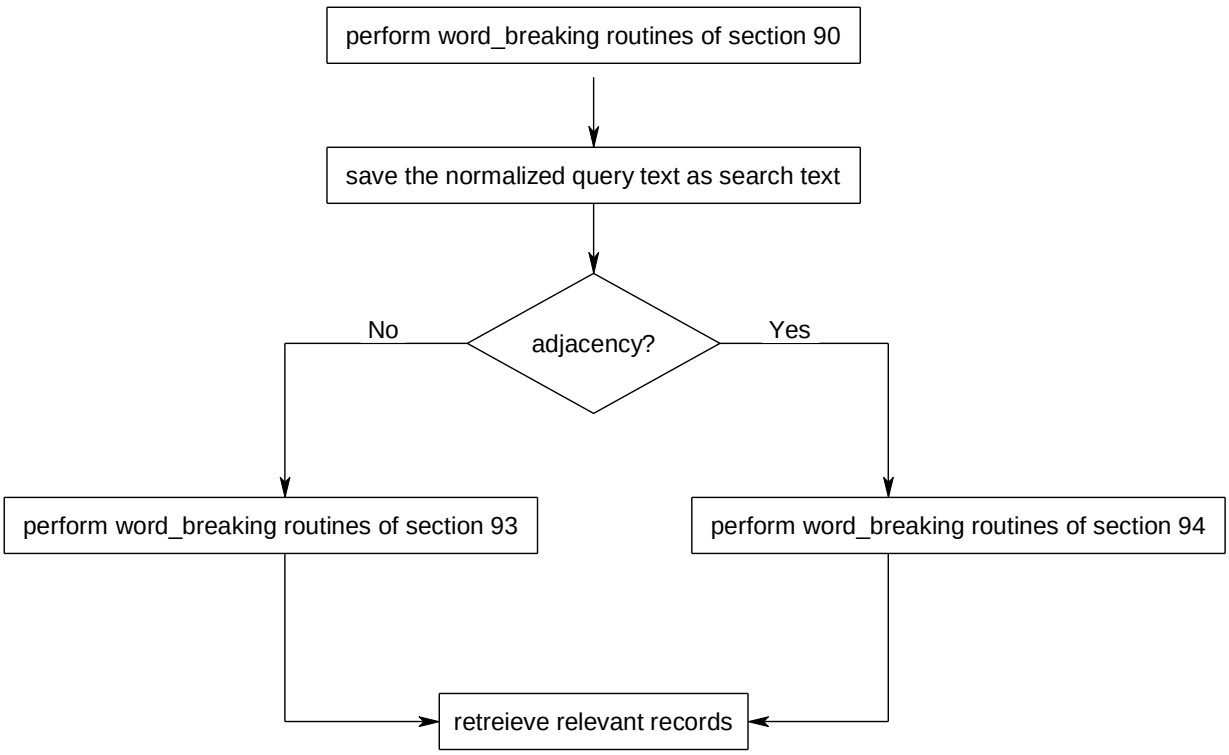
- Section 90 - the word_breaking routines set in this section will be performed for parsing a FIND query.

  The saved search text is the find query text AFTER performing the routines set in section 90.

- Section 93 – the word_breaking routines set in this section will be performed on the find query text after word_breaking routines of section 90 have been performed. The routines set in this section will be performed only when searching without adjacency.
- Section 94 – the word_breaking routines set in this section will be performed on the find query text after word_breaking routines of section 90 have been performed. The routines set in this section will be performed only when searching *with adjacency*.

Note that routines of section 90 are always performed on the FIND string, while sections 93/94 are performed only if the FIND string contains CJK text.

The following is a summary of the flow when the system segments the FIND query:



The following is an example of a tab_word_breaking setup for CJK search:

```
!1 2        3                                    4
!!-!-!!!!!!!!!!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

90 # to_blank             !@#$%^()_={}[]:";<>,.?|\

93 # char_conv            CJK_TO_NORMALIZED
93 # cjk_2gram_all

94 # char_conv            CJK_TO_NORMALIZED
94 # morpheme_search
```

Generally, it is possible to configure one word breaking routine to be used if the adjacency flag is N (section 93 of tab_word_breaking) and a different word breaking routine if the adjacency flag is Y (section 94 of tab_word_breaking).

To configure Aleph to use two different routines with different adjacency definitions:

1. Add the following to alephcom.ini:

```
[SearchFind]
FindAccurate=Y
```

This adds the Force checkbox next to the Adjacency checkbox in the GUI search tab.

The following table describes the behavior of Aleph when the Force checkbox is configured:

Adjacency Flag	Force Flag	Value to Server	Behavior
Not checked	Not checked	N	93 section + N adjacency
checked	Not checked	Y	94 section + Y adjacency
Not checked	checked	M	94 section + N adjacency
checked	checked	B	93 section + Y adjacency

2. Add M and B values to the search HTML page for OPAC.

6.2.4 Suggested APAC Indexing and Searching Segmentation Routines Setup

The following is a summary of the existing segmentation routines that handle APAC text:

- **split_cjk** – The segmentation is performed according to the Z113 table (Chinese dictionary) from left to right. In addition, the text is split, character by character.

- **cjk_to_word** – The text is divided into words from right to left according to the longest word principle by using the Z113 table (Chinese dictionary).
- **cjk_split1** – Each CJK character is considered as a word when defining segmentation routines that are performed on the search string.
- **cjk_split3** – Words are determined according to a pre-defined dictionary (z113).
- **cjk_simplified** – Characters are translated to a simplified form.
- **cjk_input_adj** – Each CJK character is considered as a word when searching with adjacency.
- **cjk_input** – Text is divided into words from right to left according to the longest word principle by using the Z113 table (Chinese dictionary). The next word starts after last character of the previous word.
- **cjk_2gram_lng** – This routine is used with a parameter. Allowed values are CONC and NO-CONC.
  - cjk_2gram_lng with parameter: CONC
    Words are:
    - All Bi-Gram segments of the concatenation of consecutive CJK substrings of the same writing system (Chinese, Hangul or Kana).
    - All space delimited non CJK substrings (“Latin Words”)
    - All space delimited single CJK characters
    - All Chinese characters
    - All Hangul characters that are a one character Korean word (defined in the /alephe/unicode/tab_cjk_single_char_word table).
    - All normalized Katakana characters that are the normalized form of one Kana character Japanese word (defined in the /alephe/unicode/tab_cjk_single_char_word table).

  - cjk_2gram_lng with parameter: NO-CONC

Words are all Bi-Gram segmentation of the CJK space delimited substrings. In other words, no concatenation of the CJK substrings is done before the Bi-Gram segmentation. In addition, all "Latin words" are considered words.

- **cjk_2gram_all** – Words are all Bi-Gram segments of the concatenation of the whole text (including non CJK text).
- **cjk_add_single** – Every CJK character is added as a word in the index.
- **cjk_add_space –** Insert space between characters of different writing systems.

- **morpheme_index –** To be used only for segmentation during the indexing process. Creates all possible substrings that are concatenation of successive words in the given field.

  The number of morphemes that will be concatenated is limited to 30. As result, if the patron will enter a search term that consists of more than 30 morphemes with no spaces between them, the search will not match any record.

  When this routine is used for segmentation during the indexing process, the cjk_morpheme_search routine must be used during the segmentation of the FIND string.
  Note that when indexing according to the Bi-Gram algorithm, Japanese characters and Chinese ideograms are handled in the same manner.

- **morpheme_search** – To be used when morphem_index is used in the indexing process.
  Note that when indexing according to the Bi-Gram algorithm, Japanese characters and Chinese ideograms are handled in the same manner.

  Note that the Morpheme index algorithm does not produce word pairs; therefore, use Morpheme search only for searching without adjacency (section 93 of tab_wrd_breaking).
- **morpheme2_index –** To be used only for segmentation during the indexing process. Creates all possible substrings that are a concatenation of successive CJK words that are less than the number of characters (configurable) in the given field.

  When this routine is used for segmentation during the indexing process, the morpheme2_search routine must be used during the segmentation of the FIND string.
- **Morpheme2_search** – To be used when morphem2_index and cjk_2gram_lng / cjk_2gram_all are both used in parallel in the indexing process. If the searched word is longer than 8 characters, this routine performs the cjk_2gram_lng algorithm. Otherwise, words are found based on the words created by the morpheme2 algorithm.

- **thai_index –** To be used only for segmentation during the indexing process.
  The text is divided into words from right to left according to the longest word principle by using the Z117 table (Thai dictionary).

  When this routine is used for segmentation during the indexing process, the thai_search routine must be used during the segmentation of the FIND string.

- **thai_search** – To be used when thai_index is used in the indexing process.
  The text is divided into words from right to left according to the longest word principle by using the Z79 table (Aleph dictionary). The next word starts after last character of the previous word.

The following options may be used for defining the indexing (in tab11_word) and searching ("93" and "94") segmentation routines:

- For indexing use *split_cjk*
  For searching use:
  - *cjk split_1*

  Or
  - *cjk split_3*

  Or
  - *cjk_char_to_simlified*
- For indexing use *cjk_to_word*

  For searching use:
  - *cjk_input_adj* (for adjacncy search)
  - *cjk_input* (for non adjacency search)
- For indexing use:
  - *cjk_2gram_lng with parameter "CONC"*
  - *cjk_2gram_all*

  For searching use:
  - *cjk_2gram_lng with parameter "NO-CONC"* (for non adjacency search)
  - *cjk_2gram_all* (for adjacncy search)
- For indexing use:
  - morpheme_index

  For searching without adjacency use:
  - morpheme_search

- For indexing use:
  - thai_index

  For searching use:
  - thai_search

- For indexing use:
  - morpheme2_index
  - cjk_2gram_lng / cjk_2gram_all

  For searching use:
  - morpheme2_search

In the following example, if the indexing is done using the "01" and "02" segmentation routines then the following tab_word_breaking setup is recommended.

```
!1 2        3                                   4
```

```
!!-!-!!!!!!!!!!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

01 # to_blank             !@#$%^()_={}[]:";<>,.?|\
01 # char_conv            HANJA_TO_HANGUL
01 # char_conv            KANA_TO_NORMALIZED
01 # cjk_add_space
01 # cjk_2gram_lng        CONC

02 # to_blank             !@#$%^()_={}[]:";<>,.?|\
02 # char_conv            HANJA_TO_HANGUL
02 # char_conv            KANA_TO_NORMALIZED
02 # cjk_add_space
02 # cjk_2gram_all


93 # to_blank             !@#$%^()_={}[]:";<>,.?|\
93 # char_conv            HANJA_TO_HANGUL
93 # char_conv            KANA_TO_NORMALIZED
93 # cjk_add_space
93 # cjk_2gram_lng        NO-CONC


94 # to_blank             !@#$%^()_={}[]:";<>,.?|\
94 # char_conv            HANJA_TO_HANGUL
94 # char_conv            KANA_TO_NORMALIZED
94 # cjk_2gram_all
```

Note that the following is required to supplement this setup:

The tab_character_conversion_line table in $alephe_root/unicode must contain the following lines:

```
!!!!!!!!!!!!!!!!!!!-!!!!!-!-!!!!!!!!!-!!!!!!!!!!!!!!!!!!!-
!!!!!!!!!!!!!!!!!
HANJA_TO_HANGUL      ##### # line_utf2line_utf             hanja_to_hangul
KANA_TO_NORMALIZED         #####       #       line_utf2line_utf
kana_to_normalized
```

The hanja_to_hangul and kana_to_normalized tables must be defined in the $alephe_root/unicode directory.

Note: If you want to index single CJK characters as words in addition to regular words, set `cjk_add_single` in `tab_word_breaking`. Note that this routine should be added AFTER the routines `cjk_to_word` and `split_cjk`, if they are in use.

7 Main Tables Supporting Indexing

The following index-related tables are explained below:

tab00.lng

tab11_acc

tab11_word

tab11_ind

tab_word_breaking

tab_filing

tab_expand

tab_character_conversion_line

tab20

tab22

tab_sort

tab_aut

tab_base

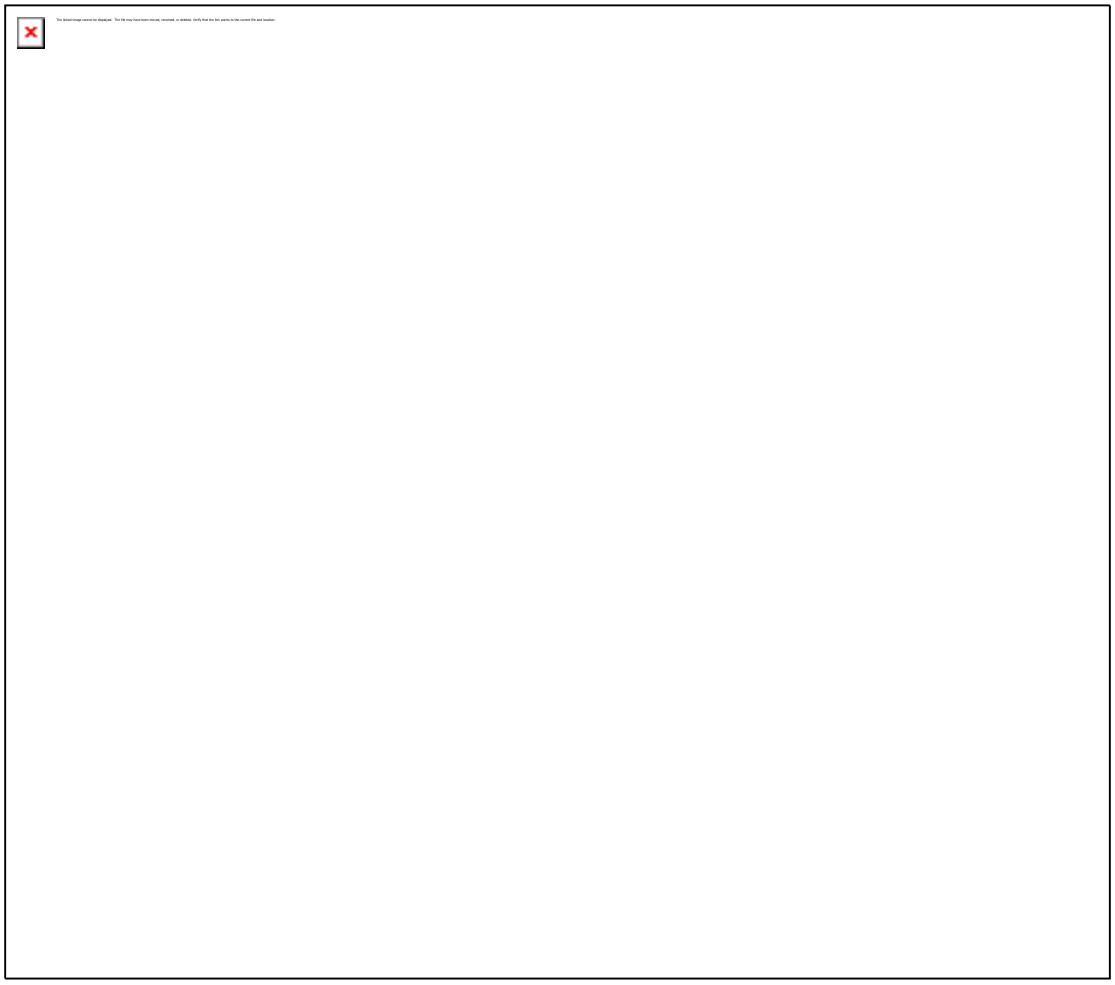
tab00.lng

This table is used to define the system access codes and names. It is divided into three sections for each type of index:

ACC file (Headings)

WRD file (Words)

IND file (Direct)



tab11_acc

The table is used to assign record fields to headings indexes. This table is limited to 1500 lines.

tab11_word

The table defines the connection between the record fields and the word indexes (one field can be indexed in one or more word groups). A field can be listed several times in tab11_word, in order to index it a number of times, with different word breaking routines each time. This table is limited to 10,000 lines.

You can define word group codes which are up to five characters in length. Note that there is no filler between the multiple word index definitions, cols. 9-18.

tab11_ind

The `tab11_ind` table is used to assign fields to direct indexes. This table is limited to 500 lines.

tab_word_breaking

This table contains word breaking specifications. For word breaking procedures, refer to that section in Sorting and Word Breaking on page 36.

tab_filing

This table is used for filing and normalization procedures that are used when building headings and for defining filing procedures that are used when building index entries and sort keys. For more information, refer to Sorting and Word Breaking on page 36.

tab_expand

Defines expand procedures which are activated when an index is created. For more information, refer to Expand Routines, Tables and Indexing Expanded Fields on page 46.

tab_character_conversion_line

Character conversion tables are used to define equivalencies for characters. All characters are sorted by their Unicode values by default. In order to force a different sort order, you can set an equivalency for sorting. The system uses the character conversion table that is listed in column 5 of the `tab_character_conversion_line` table of the `$alephe_unicode` directory

For example, the character conversion tables assigned to the WORD-FIX instance are used to define equivalencies of characters for the purpose of creating words in the words file.

tab22

The `tab22` table of the library's tab directory defines which fields will be included in the short bibliographic record. The table is also used to determine how these fields are created. The Short Bibliographic Record is built by the system, according to the definitions of this table, when records are uploaded into the database (when the indexing parameter is set to 'Full'), or when records are added or updated through the Cataloging module.

tab_sort

The `tab_sort` table defines the fields and subfields assigned to a sort key (sort keys are used for sorting a set of records in OPAC). Up to five alternative field / subfield combinations can be defined for each sort key.

Keys 02 and 03 are used by some services to define the sort by Author and by Title respectively. The actual way in which the sort by Author and sort by Title works is configurable. For example the Author can be sorted according to field 100, or field 110 or a combination of both. However, you must make sure that this configuration retains key 02 for Author and key 03 for Title.

tab_aut

Establishes which headings indexes should be subject to authority control. This table also designates - per index - which authority database is to be checked for a match.

tab_base

This table defines the logical and physical databases that can be accessed via the Web OPAC and through the Search function in the GUIs. Logical bases are defined by setting up a FIND command that serves as a pre-filter or scope. The FIND command can be up to 500 characters in length.

In order to set up a logical base that includes all records **except** for a specified group

of records, use *alldocuments* to define all records, together with *not*. For example, *alldocuments not wst=suppressed* sets up a logical base that includes all records except for those that contain *suppressed* in the *wst* word group.

tab20

Defines the rules by which a headings index is enriched from an Authority record, creating headings that are cross-referenced.

8 Sorting and Word Breaking

This chapter includes the following sections:

Sort Headings and Indexes

Sorting Item Lists

Word Breaking

8.1 Sorting Headings and Indexes

Each Heading (Z01), Index (Z11) and Sort key (Z101) record has what are termed "filing keys" or "filing text". This is the form of the heading or index term for filing (sorting) purposes. The rules that govern the values of the filing keys are set in the library's `tab_filing` table.

Headings have an additional related feature, called normalization. Normalization refers to the process whereby diacritics, most punctuation, special characters, and case differences are stripped from headings. This is in order to neutralize slight differences. This is important in the headings index where each unique heading is stored only once.

Normalization routines are defined in the `tab_filing` table together with filing routines and display text routines.

When a new heading is added to the database, if there is already another heading with the same normalized text (Z01-NORMALIZED-TEXT), even if the display text (Z01-DISPLAY-TEXT) is different, both headings are considered to be the same. In this case, no heading record (Z01) is registered for the new heading and the display text of the first heading is taken.

The filing key of the headings is built in two stages:

Display text (Z01-DISPLAY-TEXT) to normalized text (Z01-NORMALIZED-TEXT).

Normalized text (Z01-NORMALIZED-TEXT) to filing text (Z01-REC-KEY).

For this reason, when creating the filing form of the heading, it is not necessary to perform routines that have already been performed in order to create the normalized text of the heading.

The library's `tab/tab_filing` table defines the normalization and filing routines that are used. The order of the subroutines within a routine is important; for example, you

cannot relate to a subfield code if you have previously set "del subfield code". The table includes the following columns:

Column 1 - Identifier
Contains the two-digit identifier of the filing routine. This identifier is used in column 5 of the tab00.lng table (for headings and direct indexes) and column 3 of the tab01.lng table (for sort keys).

Column 2 - Routine Usage
This column is relevant only for headings (Z01). It is not relevant for direct indexes (Z11) or for sort keys (Z101). It defines the usage of the routine. The available options are:

D = Display text procedures

N = Normalized text procedures

F = Filing text procedures

Note that the routines defined in the `tab_filing` table require an 'F' section, so even if this section is not needed (because the lines for 'N' suffice), an 'F' line still needs to be present with "no" as the filing procedure.

Column 3 - Procedure
This column contains the name of the filing or normalization procedure.

Column 4 - Parameters
Parameters for the filing/normalization procedure (when relevant). If the parameters are characters, and a character is out of the ASCII range, type this character in Unicode notation.

The available filing and normalization procedures are:

abbreviation: compress a dot between single characters (for example, I.B.M. changes to IBM). This routine works only with characters in the 7-bit ASCII range.

add_prefix_hash: adds a hash (#) sign immediately after the subfield code specified in the parameters column. It is mostly required in order to correctly sort headings derived from the fix_doc_aut_duplicate program.

bbk: special procedure for Russian filing standards, in which the sorting sequence is special characters, followed by Cyrillic characters, followed by Latin characters, followed by numbers.

char_conv: perform the character conversion procedure according to the procedure name listed in col.4. This name must match procedure identification in col.1 of `/alephe/unicode/tab_character_conversion_line`.

chi: translates each character to pinyin, using the Oracle table that contains the pinyin form of Chinese characters (Z114), and adds the Unicode value for each character. The Unicode values added in order to differentiate between different characters have the same pinyin value. Since the pinyin filing-text is sequenced together with regular Latin characters, this routine should be used for browse lists that use the language code from the 008 MARC 21 field to separate by language, and are separate for Chinese (for example, AUTC). Note that separate browse lists can be created by using the expand_doc_bib_lng_cjk expand program.

cjk: adds ! before each CJK "character", and translates the characters to pinyin, using the Oracle table that contains the pinyin form of Chinese characters (Z114) and adds the Unicode value in decimal notation for each character. This causes the pinyin filing-text to be sequenced separately from regular Latin characters.

compress_blanks_cjk: Deletes space between two CJK characters.

hangul_call_no: Filing item's call number routine (column 3) for Korean libraries. The routine uses Unicode values of each character and places Hangul characters first, followed by Latin, followed by numbers. Additional value 0 or 1 can be set in column 4 of tab_filing.

0 – Default parameter, apply LC routine for hangul_call_no.

1 - Apply Dewey routine for hangul_call_no.

hangul_call_no routine includes special handling for sorting Chronology Number part of Call Number's Item part information ($$i of Z30-CALL-NO and Z30-CALL-NO-2). The numbers within the chronology number will not be sorted according to their sequential value, but according to the year they represent. This part of the sorting algorithm uses the $$9 subfield which holds the full year information.

The hangul_call_no routine uses a special entry in column 1 of tab_character_conversion_line: FILING_HANGUL_CALLNO. The library may define in column 5 of tab_character_conversion_line any character conversion table that should be used for FILING_HANGUL_CALLNO. The hangul_call_no sorting routines consult the defined character conversion table to apply the sorting routine.

The following is an example of ./alephe/unicode/tab_character_conversion_line that defines filing_hangul_callno table for FILING_HANGUL_CALLNO:

```
FILING_HANGUL_CALLNO ##### # line_utf2line_utf   filing_hangul_callno
```

The ./alephe/unicode/filing_hangul_callno table must be set to define the Unicode values to be used by the hangul_call_no sorting routine.

Each subfield of the Korean call number undergoes a different manipulation for creating the filing value of the call number. Therefore, in order to be able to search correctly, the search query must include the subfield codes.

comma: All commas are turned into blanks except for the first comma in subfield $a (particularly used for the normalization of names).

compress: Compress (i.e. strip) the characters listed in column 4. You can compress characters by specifying their Unicode values in the parameters column. In the following example, the comma (Unicode value 002C) and the period (Unicode value 002E) are compressed:

```
!1 2        3                                  4
!!-!-!!!!!!!!!!!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!->
01 N compress             U+002CU+002E
```

compress_blank: delete blanks

compress_perblnk: compresses a blank when it is preceded by a period ("."). This is needed for the filing of certain LC Call Numbers.

del_lead_space: deletes any leading spaces after a subfield. For example: $$a XXX is changed to $$aXXX. The database data never has a space after the subfield code, but

the to_blank routine could create a space. Accordingly, the del_lead_space routine must be placed before del_subfield.

del_subfield: delete subfield sign ($$x)

del_subfield_code: the "$$" sign is retained, but the subfield code is replaced by a hyphen ("-"). This is used for normalization, so that headings will match when the subfield content is the same, even if the subfield codes are different.

dewey_call_no: special procedure for the correct sequencing of Dewey Call Numbers.

end_punctuation: deletes the characters listed in column 4 of the `tab_filing` table when this is the last character in the heading. Note that this function is mostly used to remove / : = and so on at the end of a title, and so on. It is intended for routines of type "D" - display text conversions.

end_sub_punctuation: This routine deletes the characters listed in column 4 of the tab_filing table from each subfield comprising the heading.

expand_num: expand number. This routine adds leading zeroes to fill numbers to a certain number of digits for numeric filing. The maximum number of digits can be specified as a one or two digit number by using the parameters column in `tab_filing` (col. 4). If column 4 is left blank, then a default of 7 digits is used.

Following is a sample of the `tab_filing` table:

```
!1 2        3                                      4
!!-!-!!!!!!!!!!!!!!!!!!!-
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

11 F expand_num           13
```

get_subfields: use only the subfields, or subtract some using "-" as listed in col.4.

get_subfields_order: this procedure is similar to the get_subfields routine except that it retains the order of the subfields specified in col. 4.

icelandic_name: changes the order of subfields 7 and 1, placing subfield 7 after subfield 1. Intended for sorting OPAC browse lists. For example, the display text:

```
$$aAlexander $$7Alfred $$1Jonsson$$c1943-
```

becomes the filing text:

```
Alexander Jonsson Alfred 1943-
```

jpn: translates each character to the decimal value of the Unicode character. This causes the filing-text to be sequenced together with regular characters. Therefore, this routine should be used for browse lists that use the language code from the 008 MARC 21 field to separate by language, and that are separate for Japanese (for example, AUTJ). Note that separate browse lists can be created by using the `expand_doc_bib_lng_cjk` expand program.

kor: translates each character to the decimal value of the Unicode character. This causes the filing-text to be sequenced together with regular characters. Therefore, this routine should be used for browse lists that use the language code from the 008 MARC 21 field to separate by language, and for those that are separate for Korean

(for example, AUTK). Note that separate browse lists can be created by using the `expand_doc_bib_lng_cjk` expand program.

lc_call_no: special procedure for correct sequencing of LC Call Numbers. Note that this routine adds the following three characters to the index records it creates: ! " #. For this reason, you cannot have a to_blank or compress line which includes these characters after the lc_call_no line. In addition, note that this procedure is complete within itself, and does not require additional treatment. However, in order to facilitate searching, it is recommended that del_subfield be added.

lc_call_no_2: same as lc_call_no, except that this routine enables non LC call numbers to be included in call number indices.

mc_to_mac: change initial mc to mac

no: this routine is used when the procedure defined in the `tab_filing` table does not contain an 'F' section. The procedures defined in the `tab_filing` table require an 'F' section, so even if this section is not needed (because the 'N' section - used for normalization purposes - suffices), an 'F' line still needs to be present with "no" as the filing procedure. For example:

```
01 F no
01 N to_blank              !"()-{}<>;:.?/\@*%=^_`~
01 N comma
01 N del_subfield_code
01 N char_conv             FILING-KEY-10
```

non_filing: drop initial text using non-filing indicator

non_numeric: delete non-numeric characters

none: This routine builds the call number key as the call number itself.

numbers: compress a comma and a dot between numbers (for example, 2,153 changes to 2153)

pack_spaces: compresses all multiple spaces to a single space

subfield_mab: intended for filing of headings which are based on MAB-authority information. In order to exclude the identification number from sorting, the procedure adds three blanks at the beginning of each subfield, from the second subfield on, and adds four blanks to the beginning of $$9. Subfield codes are removed.

suppress: this program drops all text contained within the signs << and >>, including the characters themselves. You can also add comma-delimited parameters to the *suppress* routine in Column 4 of `tab_filing`. Here is an example from `tab_filing`:

```
!1 2        3                                        4
!!-!-!!!!!!!!!!!!!!!!!!!-
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
94   suppress             88-89,<<>>
```

If the parameter *88-89* is specified, the control characters U+0088 and U+0089 will be used instead of << and >>. The parameter <<>> acts the same way as the default.

If both parameters are specified, the input text will undergo suppression twice: once with << and >> as delimiters, and again - with U+0088 and U+0089 as delimiters (or vice versa).

The following parameter combinations are allowed:

<<>> The same as the default

88-89 Suppression only with U+0088 and U+0089 as delimiters.

88-89,<<>> (2) and then (1)

<<>>,88-89 (1) and then (2)

to_blank: change characters listed in col.4 to blank.

You can compress characters by specifying their Unicode values in the parameters column. Enter the notation in the form U+<Unicode value> - for example "U+0153".

to_blank_2: changes to blank the characters listed in the parameters column (col. 4). The character is changed to a blank only if it is followed by a blank or if it is at the end of the field.

For example, if the comma is listed in the parameters column under this procedure, then "Schiller, Friedrich" is changed to "Schiller Friedrich", but "one,two,three" is not changed.

to_lower: change to lowercase

to_carat: change subfield sign to ^^ (for hierarchical sorting of headings). WARNING! Although this will file a heading such as "$$aArt $$zZambia" before "$$aArt, Canadian", when the system performs a browse search, the search query is taken word-by-word, character-by-character, transforming multiple blanks to a single blank. Therefore, it is not possible to zero in on "art ^^zambia" in a browse search, and although the list will be hierarchically arranged, it will be difficult to use.

year_uu: replaces *u* with zero (*0* in the year formats where a date element is unknown, for example, 19uu, 197u. Note that it should be activated before the **expand_num** routine.

Following is a sample of the `tab_filing` section used for title headings:

```
11 D end_punctuation      :,=;/
11 N to_lower
11 N to_blank             !@#%^&*()_+-={}[]:";?,./~`
11 N pack_spaces
11 N del_subfield_code
11 F del_subfield
11 F suppress
11 F numbers
11 F to_blank             $<>
11 F expand_num
11 F non_filing
11 F compress             '
11 F pack_spaces
11 F char_conv            FILING-KEY-01
11 F cjk
```

8.2 Sorting Item Lists

The sorting order of items throughout most modules of the system is determined in a single table, `tab_z30_sort` in the Administrative library.

This table includes sort options for both issue and non-issue type items for each module/function which includes lists of items.

The sort options are made up of two elements:

Sorting type - defines the various levels of sorting. For example, serial items can be sorted by volume, then by issue number, then by part number.

Sorting order - defines whether the sorting will be ascending or descending.

The following table includes the modules/functions which are dealt with by this table and their codes:

Module / Function	**Code**
Web OPAC	WWW-A
Web Course Reading	WWW-R
Serials client	SERIAL
Search client	SEARCH
Circulation client	CIRC
Items client	ITEM
Items for binding	ITEM-BIND
Acquisitions/Serials client	ACQ
Services	BATCH
Lost Item Report	CIR-16
for use by fix_doc_create_86x procedure	86x
Navigation window	TREE

8.2.1 Sort Options

The available sorting types are:

For **issue type** items:

00 - if chronological-i(year) is spaces and enumeration-a(volume) is spaces, then description + item-sequence. if chronological-i(year) is not spaces, then chronological-i(year) + enumeration-a(volume) + enumeration-c(part) + enumeration-b(issue) + item-sequence.

01 - if chronological-i(year) is spaces and enumeration-a(volume) is spaces, then description + item-sequence. if chronological-i(year) is not spaces, then chronological-i(year) + enumeration-a(volume) + enumeration-b(issue) + enumeration-c(part) + item-sequence.

02 - if chronological-i(year) is spaces and enumeration-a(volume) is spaces, then description + item-sequence. if chronological-i(year) is not spaces, then hol-doc-

number + chronological-i(year) + enumeration-a(volume) + enumeration-b(issue) + enumeration-c(part) + item-sequence.

03 - sublibrary + item-sequence.

06 - sublibrary + collection code + chronological-i(year) + chronological-j (year) + chronological-k(year) + description + copy-id (site specific).

07 - 85x-type + sublibrary + collection + linking-number + if enumeration is not spacesi, then enumeration. if enumeration is spaces, then description.

08 - 85x-type + sublibrary + collection + linking-number + copy-id + if 85x-type is 4 or 5 then supp-index-o + if chronological is not spaces and enumeration is not spaces, then chronological + enumeration.

12 - sub_library + collection + if chronological-i(year) is spaces and enumeration-a(volume) is spaces then description + if chronological-i(year) not spaces then chronological-i(year) + enumeration-a(volume) + enumeration-b(issue) + enumeration-c(part) + copy.

13 - by barcode (using filing routine code 98)

For **non-issue type** items:

00 - if enumeration-a(volume) is not spaces, then: enumeration-a(volume) + enumeration-b + enumeration-c(part) + sublibrary + collection. if enumeration-a (volume) is spaces, then: description + enumeration-c(part) + sublibrary + collection.

01 - enumeration-a(volume) + enumeration-b + enumeration-c(part) + description + sublibrary.

02 - if enumeration-a(volume) is not spaces, then: enumeration-a(volume) + enumeration-b + enumeration-c(part) + sublibrary. if enumeration-a(volume) is spaces, then: description + sublibrary.

03 - if description is blank, then: enumeration-a(volume) + chronological-i (year) + enumeration-b + enumeration-c(part) + sublibrary + item status. if description is not blank, then: description + sublibrary + item status.

04 - if enumeration-a(volume) is not spaces, then: hol-doc-number + enumeration-a(volume) + enumeration-b + enumeration-c(part) + sublibrary + collection. if enumeration-a(volume) is spaces. then: hol-doc-number + description + sublibrary + collection.

05 - sublibrary + item-sequence.

06 - sublibrary + collection code + description + copy-id (site specific).

12 - sub_library + collection + if chronological-i(year) is spaces and enumeration-a(volume) is spaces then description + if chronological-i(year) not spaces then chronological-i(year) + enumeration-a(volume) + enumeration-b(issue) + enumeration-c(part) + copy.
13 - by barcode (using filing routine code 98)

The available sorting orders are:

A - Ascending

D - Descending.

For the Items, Serials and Circulation modules, you can define more than one sorting option in tab_z30_sort, but each option must be given its specific code (for example, CIRC-1, CIRC-2, and so on).

These options must then be also defined in the `pc_tab_exp_field.lng` table of the Administrative library, so that they will be available as drop-down menu options in the Item List windows of these modules.

8.3 Word Breaking

For word indexing, individual words of a field are written in the words table. Basically, a word is a group of characters between white spaces. The library's `tab_word_breaking` table is used to define instances where other factors are taken into account. For example, is a hyphen a space, or is a hyphen compressed and treated as if it were not there at all?

Text undergoes word breaking using the characteristics that are listed in one routine in the `tab_word_breaking` table. The structure of the table includes four columns:

col.1: Two-digit identifier of the word breaking routine. This identifier is used in column 6 of tab11_word.

col.2: Used only for identifying CJK range.

col.3: Name of the word breaking procedure

col.4: Parameters for the word breaking procedure (when relevant)

Facets for the word breaking procedures are:

2_hyphen: this routine changes two adjacent hyphens (--) to a blank. This word breaking procedure can be used, for example, to change to blank consecutive hyphens in the MARC 21 field 505 (FORMATTED CONTENTS NOTE) that uses hyphens as separators. The following is an example of a 505 field:

```
5050  L $$aHow these records were discovered -- A short sketch
of the Talmuds -- Constantine's letter.
```

compress: compress (that is, strip) the characters listed in col.4

compress_blank: delete blanks

del_subfield: change subfield sign ($$x) to blank

Force_delimiter: This routine changes the subfield sign ($$x) to blank-z-blank in order to prevent words across subfields being considered adjacent.

to_blank_2: change characters to blank if the character is followed by a blank

to_blank: change characters listed in col.4 to blank

subf_to_sign: change second and subsequent subfield signs to the single character listed in col.4

ccl_brackets1: This routine searches for a term that includes parenthesis (brackets) in a word. The parameter of the routine "ccl_brackets1" can be either "c" (compress brackets) or "b" (replace by blank). If the routine "to_blank" is used before

"ccl_brackets1", make sure that the parenthesis are not included; otherwise, "to_blank" removes all parenthesis (brackets).

blank_to_carat: change blanks to caret (^)

numbers: compress a comma and a dot between numbers, for example, 100,000 or 100.000 -> 100000

abbreviation: compress a dot between single characters. For example, I.B.M. becomes IBM.

marc21_41: used for separating in MARC21 041 field

Notes:

The procedures must be listed in logical order. For example, numbers must be listed before compress or change_to_blank if a comma or a dot are included in them. Otherwise, they will no longer be present when the numbers procedure is used.

Word breaking procedures are defined in the tab11_word table (column 6) of the library's tab directory. A line can be listed several times in the tab11_word table in order to index it multiple times, with different word breaking procedures each time.

In the following example, words from the 100 field are indexed according to the word breaking procedures 01 and 02:

```
100##                   -6                   01     WRD  WAU
100##                   -6                   02     WRD  WAU
```

In `alephe/unicode`, there is a table called `unicode_to_word_gen`. The system automatically uses this table when building the word breaking. The table can (and does) include values that change a character to a blank (by assigning the value 0020) or can compress a character (by assigning the value 0000). When browsing a word index in the OPAC, special characters are always displayed in their converted state. So, if the `unicode_to_word_gen` table sets umlaut to ue, the word is displayed with ue, and not with an umlaut.

Note that the system automatically carries out triple posting for hyphens and apostrophes: (1) as separate words; (2) as is (with hyphen/apostrophe); (3) with hyphen/apostrophe compressed.

For example: twenty-five is indexed as:

twentyfive

twenty

five

twenty-five

The "hyphen" and the apostrophe must be left with their actual value in the `alephe/unicode/unicode_to_word_gen` file, and both the hyphen and the apostrophe must not be entered in any of the word breaking procedures in the library's `tab/tab_word_breaking` file.

Use procedure 90 in `tab_word_breaking` when parsing a Word search query.

Use procedure 97 in `tab_word_breaking` in the ILL library for parsing records during the ILL Locate process.

9 Expand Routines, Tables and Indexing Expanded Fields

This chapter includes the following sections in order of appearance:

Expand Record

Expand Routines

Expand-Related Tables

Indexing Expand Fields (Virtual Fields)

9.1 Expand Record

The ALEPH integrated library system holds information in different types of records and in different types of "libraries" (databases).

A standard system has a BIB library for storing bibliographic data and an ADM library for storing administrative data. Most installations will also have an AUT library for storing authority information and a HOL library for storing holdings and location information.

Libraries may want to display information from the non-bibliographic databases together with the linked bibliographic record. Libraries may also want to enable the user to search the bibliographic database using information from other databases, such as location.

ALEPH enables the installation to "expand" information from one database record to another. This is possible because there are links between the records in the various databases. The information that is "expanded" can be used for display and/or for indexing. "Expand" routines can also be used to "expand" data within a record.

The "expand" function works with the tab_expand table located in the library's tab directory. Every library has such a table, although it is the bibliographic library that uses the table the most.

The tab_expand table defines three aspects:

The system function in which the expand program works.

The expand program that defines which data from the record can be expanded.

Additional parameters for the expand program, if required.

Following is a sample of the `tab_expand` table:

```
!   1                   2                                   3
!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
U39-DOC    expand_doc_fmt
U39-DOC    expand_doc_join
```

Key to tab_expand table:

Column 1 - System context
This column contains the "context" in which the expand is operative.

The available "context" functions are:

ACC
Context: Headings index creation/update (p_manage_02).

BIBHOL-MAB

Context: Use in the holdings library in MAB format with `expand_doc_bib_hol_mab`. Displays holding expanded fields in the bibliographic record.

BUF-Z403

Context: Used to enrich the BUF-Z403, which is used to govern access to electronic resources. This is required when, for example, expanding the 856 field, or any other field with electronic contents, into the holdings library. Note that expand_doc_bib_z403 expand procedure may not be used with expand menu BUF-Z403.

CREATE-Z13
Context: Short bibliographic record creation/update (p_manage_07).

E-DOC-<format number>
Context: Specific format display. Used for running expand programs that should be applied only to specific formats. For example, the `expand_doc_uni_merge` program should be functional only when the record is displayed in ISBD format.

The format number of the instance should match the format number defined in the `edit_doc.lng` table for the desired format. For the `expand_doc_uni_merge` example mentioned above, if the ISBD format has been defined as 038 in the `edit_doc.lng` table, then the `tab_expand` table should be defined as follows:

GUI-ACCREF
Context: Authority record display from bibliographic heading (Search module).

GUI-BRIEF
Context: Brief display (Search module).

GUI-DOC-D
Context: Full display (Search module).

GUI-DOC-P
Context: Full print (Search module).

HOL-LOC
Context: Use in the holdings library with `expand_doc_hol_loc_1_a` and

`expand_doc_hol_loc_2_a`.

HOLDING
Context: Display of item list.

INDEX
Context: Direct index creation/update (p_manage_05).

PRE-MERGE

Context: Adds expanded fields to the merged display of a record in Union catalog/view

PRINT-CAT
Context: Print catalog (p_print_04).

PRINT-CUST
Context: Print custom format (p_print_01).

PRINT-COL
Context: Print columnar format (p_print_08).

PRINT-REC

Context: Print Catalog Records with "Non-preferred" Headings (print-05).

RET
Context: Retrieval of records (p_ret_01) and sorting (p_ret_21).

SORT-DOC
Context: Sort keys creation/update (p_manage_27).

U39-DOC
Context: Record display through UTIL F/4/DOC.

UE-08
Context: UE-08 (for expanding authority records for UE-08 procedures).

WEB-ACCREF
Context: Authority record display from bibliographic heading (Web OPAC).

WEB-BRIEF
Context: Brief display (Web OPAC).

WEB-FULL
Context: Full display (Web OPAC).

WEB-FULL-1
Context: Full display - format 01 (Web OPAC).

WEB-MAIL
Context: Full print - mail (Web OPAC).

WEB-SCNIND
Context: Title display when browsing Direct indexes (Web OPAC).

WORD
Context: Word index creation/update (p_manage_01).

X-AVAIL

Context: Retrieves the current availability status of a document (can be expanded using "expand_doc_bib_avail" and parameters in Col.3 of tab_expand).

Z00R
Context: Creation of a Z00R record (p_manage_07, cataloging, UE_01).

Z39-SERVER
Context: Z39-SERVER.

Column 2 - Expand program
The expand program that defines which data is expanded.

Column 3 - Parameters
Certain expand programs require additional information, such as field codes. This column is used to define additional parameters for expand programs. Note that the documentation for each expand program indicates whether or not parameters are needed (see for example, expand_doc_sort_field).

9.2 Expand Routines

The following are the available expand routines:

Note that some expand programs have suffixes like *usm* and *mab*. This convention is used for expand programs dependent on the MARC format (such as MARC21, MAB, UNIMARC, and so on).

---

expand_doc_acronym_title

The expand_doc_acronym_title routine facilitates acronymic indexing of long titles that have common words (for example, Report of the …Association). This expand creates a new index which will help users search more easily for serial titles. It takes the first four letters from the first word of the title connected to the first three letters of the second word, connected to the first two letters of the third word, connected to the first letter of the fourth word. If the words are short, only the existing letters will be used. For example: The title “Gen. hosp psych” will be indexed as genhosps.

Stop words such as "the”, "and”, or "to", which reside in tab03 in the bibliographic library are not considered when building the "Acronymic Title".

This expand should be used when WORD indexing is performed.

Column 3 needs to be defined to include the record format, field and subfield to index and the tab_word_breaking procedure that will be used.

```
!   1                   2                              3
!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!>
WORD       expand_doc_acronym_title       SE,245##a,92
WORD       expand_doc_acronym_title       BK,245##a,92
```

This expand should use a tab_word_breaking procedure such as the following:

```
  92 # del_subfield
  92 # to_blank             !@#$%^()_={}[]:";<>,.?|\
  92 # compress             '
  92 # to_lower
```

Note that the word breaking routine number in column 1 is definable.

In this following example, a word index was created to support searching/browsing:

tab11_word:

```
! 1      2    3       4              5               6   7 8   9    10    11
12
!!!!!-!!!!!-!-!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!-!!-!-!-
!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!
ACT##                      a                      92      WAC
```

tab00.lng:

```
!   2      3    4 5  6  7 8    9   10             11
!-!!!!!-!!!!!-!-!!-!!-!-!---!!-!!!!!-!!!!!!!!!!!!!!!!!!!
H WAC    W-026       00        00        W-act.
```

expand_doc_adm_bib

The `expand_doc_adm_bib` program adds bibliographic data to the administrative record.

To include or exclude specific BIB fields in the expanded ADM record, use column 3 of `tab_expand` of the ADM library.

The parameters in Column 3 of `tab_expand` are up to 10 comma-separated fields (five characters each, can include hashes, for example 100##), to be included or excluded in the ADM record.

To exclude specific fields, the list of fields to be excluded should be preceded by a dash - , for example -245##,100##.

If column 3 is blank, then ALL fields from the BIB record are added to the ADM record.

In the following example, the 260## and 100## fields will be excluded from the expanded ADM record

```
!   1                    2                                3
!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!>
U39-DOC    expand_doc_adm_bib             -260##,100##
```

Note that this expand program must be defined in the `tab_expand` table of the administrative library (XXX50). This can be useful for creating short_doc (Z13) (using manage-07) in the ADM library.

Note that this expand routine's behavior is the same as expand_doc_hol_bib.

expand_doc_adm_hol

The `expand_doc_adm_hol` expand routine is used in the `tab_expand` table of an ADM library in order to add HOL data to the ADM record.

Parameters in column 3 of tab_expand can be up to 10 comma-separated fields (five characters each, can include hashes, for example, 85###). If column 3 is blank, then all fields from the HOL record are added to the ADM record.

To exclude a specific field, precede the list of fields with a dash - , (for example, -85###, -86###).

Example of `expand_doc_adm_hol` in `tab_expand` (ADM library):

```
!   1                    2                                3
!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!>
U39-DOC    expand_doc_adm_hol              85###,86###
```

If "MERGE-TYPE=" is used in Column 3 of `tab_expand` for `expand_doc_adm_hol`, it is possible that more than one HOL record will be merged into the ADM record. For example, in `expand_doc_adm_hol`, if more than one HOL record is linked to the ADM record, the first HOL record is merged into the ADM record and then the second HOL record is merged into the already merged ADM record, and so on.

Note that this expand routine's behavior is the same as expand_doc_hol_bib.

expand_doc_aut_aut

For multilingual applications, the `expand_doc_aut_aut` program identifies the authority record of a heading that is a "see also" in the authority record. The program adds all forms of the heading from the "main" authority record. This program builds the "See also" field for all languages for Broader, Narrower, and "See also" terms.

expand_doc_bib_001

The `expand_doc_bib_001` program builds a 001 field that contains the system number of the record. The field is built only if the 001 field does not already exist in the record.

expand_doc_bib_852_1

The `expand_doc_bib_852_1` program expands the 852 MARC21 location field into the bibliographic record. The field is brought from the holdings record and/or built from the information in the Z30 (item record). If the holdings record has an 866 MARC21 field (textual holdings statement), the field is appended to the 852 field from the holdings record that is expanded into the bibliographic record.

Sublibrary (subfield $b) and collection (subfield $c) codes are expanded into subfields $4 and $5 in which the sublibrary code and collection code are replaced by names using the `tab_sub_library` (sublibrary definitions) and `tab40` (collection definitions) tables. Items barcode are expanded into 852$$p.

The second call number (Z30-CALL-NO-2) is expanded using the same subfield definitions used for expanding the regular call number (Z30-CALL-NO), but in uppercase. For example:

```
852  $$bULINC$$cGEN$$HHG939.5 D38 1970$$bLincoln
Library$$cGeneral$$HHG939.5 D38 1970
```

You can define the field, subfield and subfield contents to filter the holdings records that are expanded. This can be done by defining the field, subfield and subfield contents in the parameters column (col. 3) of the library's tab_expand table. For example, if the tab_expand table contains the following line:

```
PRINT-REC  expand_doc_bib_852_1           852##,b,ULINC
```

When the holdings information is expanded into the bibliographic record, the holdings data is included only if subfield $b of the 852 field contains the value 'ULINC'. Holdings records that do not match this definition are not included.

The format for the filtering definitions is the following:

```
FIELD,Subfield,CONTENTS
```

To prevent item barcodes from being expanded into field 852 when using expand_doc_bib_852_1, use the "BARCODE=N" parameter.
Note that this parameters must be sent as the fourth parameter, even if the other three parameters are not used.

For example:

```
!   1                   2                              3
!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
GUI-DOC-D  expand_doc_bib_852_1           ,,,BARCODE=N
GUI-DOC-D  expand_doc_bib_852_1           852##,b,WID,BARCODE=N
```

expand_doc_bib_852_title

The `expand_doc_bib_852_title` program expands the subfields $a, $b, $c of the Main Title field (245 MARC21 field) into the 852 MARC21 field ($$a, $$b, $$c) and creates a new CLN field. For example, if the bibliographic record contains the following fields:

```
24510 L $$aWall shadows; :$$ba study in American prisons,
852   L $$bUELEC$$hhE183.8.B7$$iL494
```

then the `expand_doc_bib_852_title` program creates the following new CLN field:

```
CLN01 L $$hhE183.8.B7$$iL494$$1Wall shadows; : $$2a study in
American prisons,
```

If there are multiple 852 fields, then each one is taken and the same 245 field is added to each.

expand_doc_bib_880_n

The `expand_doc_bib_880_n` program creates two new concatenated fields out of 2 fields linked by subfield $6. Subfield $6 contains data that links fields that are different script representations of each other. The two new fields are concatenated once as [romanized form] + [vernacular form], and once as [vernacular form] + [romanized form]. Concatenation is subfield by subfield and subfield codes are retained, differentiated by lowercase and uppercase. For example, if the record has the following linked fields:

```
245 10 $601$aSosei to kako$bNihon Sosei Kako Gakkai shi.

245 10 $601$1$a<Title in Japanese script>$b<Subtitle in
Japanese script>.
```

then the `expand_doc_bib_880_n` program creates the following two new virtual fields:

```
245 10 $$603$$aSosei to kako$$A<Title in Japanese
script>:$$bNihon Sosei Kako Gakkai shi.$$B<Subtitle in
Japanese script>.

245 10 $$603$$a<Title in Japanese
script>$$ASosei to kako$$b<Subtitle in
Japanese script>.$$BNihon Sosei Kako Gakkai shi.
```

expand_doc_bib_accref

The `expand_doc_bib_accref` program adds non-preferred terms to the bibliographic record in order to build word entries from cross-references. This feature allows the user to perform a "find" search on preferred or non-preferred terms with the same result.

The `expand_doc_bib_accref` should only be used with the WORD system function.

If ADDITIONAL-INFO is sent as a parameter to `expand_doc_bib_accref`, the following additional information is added for each heading (the original authorized field and the added terms) in all formats (MARC21, MAB, UNIMARC and DANMARC):

Aleph Authority Record ID:

- If a related authority record exists in Aleph, a subfield of the authority record ID is added in the following format:

```
<Library><doc number>
```

For example:

```
USM10000000123
```

- The subfield where the authority record ID is added is different for each format:
    - MARC format: the authority record ID is added in subfield 0
    - UNIMARC format: the authority record ID is added in subfield 3
    - MAB format: the authority record ID is added in subfield 9
    - DANMARC format: the authority record ID is added in subfield I
- If the original field already has an authority record ID stored in a subfield as described above, it is removed.

preferred/non-preferred indicator:

- If related authority record exists in Aleph, subfield P is added with an indication if the term is the preferred term or not:
    - For non-preferred terms, the value of subfield P is N
    - For preferred terms, the value of subfield P is Y
- If no related authority record exists in Aleph, subfield P is not added.

expand_doc_bib_accref_1

This expand program works like expand_doc_bib_accref. The difference is that the cross reference information expanded by expand_doc_bib_accref_1 is put into lines named after the acc code of the relevant Z01 record of the bibliographic library.

If, for example, the acc code of the relevant Z01 record is AUT then the line to which the information is expanded is called: AUT. The expand_doc_bib_accref_1 should only be used with the WORD system function.

expand_doc_bib_adm

This expand program takes the ADM record fields and expands them in the connected bibliographic record.

Note that the expand works only in a single ADM environment.

expand_doc_bib_avail

The expand program, expand_doc_bib_avail, brings items and holdings availability information. This program relies also on the special consortia item record (z300).

The expanded information is presented in 'AVA' field which has the following sub fields:

$$a ADM library code

$$b Sub library code

$$c Collection text – If there are several items in different collections in one sub library, only the first collection of the sub library is presented.

$$d Call number – If there are several items in different collections in one sub-library, only the first call number in the collection is presented.

$$e Availability status – Can be 'available', 'unavailable', or 'check_holdings'. Available status is assigned if the total number of items minus unavailable items is positive. Unavailable is assigned if the total number of items minus unavailable items is zero or negative. If a record has no linked items (only Holdings records) the status is 'available'. This subfield may be affected by the value of the THRESHOLD parameter in column 3 of tab_expand.

$$f Number of items (for the entire sub library not just location).

$$g Number of unavailable items (for the entire sub library not just location).

$$h Multi-volume flag (Y/N) – If first item's Z30-ENUMERATION-A is not blank or 0 then =Y otherwise = N.

$$i Number of loans (for the entire sub library not just location).

$$j Collection code of the item.

$$k including the call number type as following:

- If the AVA field is built from holding information:
    - If the first indicator of field 852 in the holdings record is 7, the value of 852 subfield $$2 is copied to AVA$$k.
    - If the first indicator of field 852 in the holdings record is not 7, it is copied to AVA$$k.
- If the AVA field is built from item information:
    - If the item's call number type (Z30-CALL-NO-TYPE) is 7, the value of subfield $$2 of the item's call number (Z30-CALL-NO) is copied to AVA$$k.
    - If the item's call number type (Z30-CALL-NO-TYPE) is not 7, it is copied to AVA$$k.

Note that the above description is applied for all the formats (MARC, UNIMARC, DANMARC, and MAB).

$$p A number that represent the priority of the item by its location. The expand_doc_bib_avail routine consults ./bib_lib/tab/ava_location_priority and AVA$$p is created with a number that represents the location priority.

ava_location_priority list sublibrary and collection by their priority (the items at the top of the list have higher priority).

If there is no match with the ava_location_priority table, no subfield p is created.

$$t Contains a translation of the content of subfield 'e' according to the text of the messages entered in table ./errror_lng/expand_doc_bib_avail.

An item is unavailable if it matches one of the following conditions:

- It is on loan (it has a Z36).
- It is on hold shelf (it has Z37_status=S).
- It has a processing status and does not have a value in Z30-DEPOSITORY-ID.

Items with process statuses are considered ‘unavailable’. Note that Col.3 of tab_expand can be used to specify item process statuses that their items should be treated as ‘available’.

To ignore item process statuses that should be treated as ‘available’, set the following parameter in column 3 of tab_expand: AVA=BD,MK. It is possible to specify more than one process statuses, delimited by a comma “,”.

If you want to take reshelving time into account; set the following parameter in column 3 of tab_expand: RESHELVING=Y. This is mostly relevant for Real Time Availability functionality (availability X service).

To retrieve the availability information by collection (in addition to sublibrary), set the following parameter in column 3 of tab_expand: COLLECTION=Y.

To define the maximum number of items to check per sublibrary, set the following parameter in column 3 of tab_expand: THRESHOLD=080. In this example, the maximum number of items per sublibrary is 80. Note that the number set in this parameter must have three digits. This parameter affects the content of subfield $$e (Availability status) and sub field $$t (translation of $$e).

expand_doc_bib_avail_hol

The expand program, `expand_doc_bib_avail_hol`, brings holdings and item availability information, based on the Holding records 852 field and subfields.

For each HOL record, an AVA line is created with the holding information and its availability.

The expanded information is presented in the AVA field which has the following sub fields:

$$a ADM library code

$$b Sub library code, based on the 852$$b of the HOL record

$$c Collection text, based on the 852$$c of the HOL record

$$d Call Number - The HOL record’s 852 subfields which are set in aleph_start variable: correct_852_subfields (can be 1 or more of the following subfields: hijklm)

$$e Availability status - Can be “available” or “unavailable”, “check_holdings” or “temporary_location”. Available status is assigned if the total number of items minus unavailable items is positive. Unavailable is assigned if the total number of items minus unavailable items is zero or negative. If a record has no linked items (only

Holdings records) the status is “check_holdings”. If all the items linked to the Holding records are in a temporary location, the status is “temporary_location”. This subfield can be affected by the value of the THRESHOLD parameter in column 3 of tab_expand.

$$f The number of Items that are linked to the HOL record. If no items are linked to the HOL record, it is set to 0.

$$g The number of unavailable items (for the entire sub library not just location) that are linked to the HOL record. If no items are linked to the HOL record, it is set to 0.

$$h Multi-volume flag (Y/N) – If the first item’s Z30-ENUMERATION-A is not blank or 0 then =Y otherwise = N.

$$i The number of loans (for the entire sub library not just location). Based on the HOL record’s linked items. If no item is linked, it is set to 0.

$$j Collection code (852$$c of the HOL record).

$$k Call Number type 1st indicator of 852. If the first indicator of field 852 in the holdings record is 7, the value of 852 subfield $$2 is copied to AVA$$k.

$$p Location priority. A number that represent the priority of the item by its location.

The expand_doc_bib_avail_hol routine consults ./bib_lib/tab/ava_location_priority and AVA$$p is created with a number that represents the location priority.

If there is no match with the ava_location_priority table, no subfield p is created.

If there are 2 HOL records, for example, with the same sublibrary + collection values, then two AVA$$p subfilelds are created with the same priority rank.

$$t Availability text translation. Contains a translation of the content of subfield 'e' (available status), according to the text of the messages entered in table ./errror_lng/expand_doc_bib_avail (the same one as used by expand_doc_bib_avail).

$$7 Holdings ID - Contains the HOL library code and the HOL record number (e.g. USM60000000741) for which the AVA is created. Non-relevant for AVA fields of items with no linked HOL record.

Routine Additional Parameters

To define additional subfields added to the AVA fields from the 852 field of the HOL, set the parameter `SF` in column 3 of tab_expand (for example: `SF=z,t`) to copy subfields $$z and $$t. Note that the additional subfields that are copied from the HOL record 852 to the AVA field override the subfields created this expand program.

To define the maximum number of items to check per sublibrary, set the following parameter in column 3 of tab_expand: `THRESHOLD=080`. In this example, the maximum number of items per sublibrary is 80. Note that the number set in this parameter must have three digits. This parameter affects the content of subfield $$e (availability status) and sub field $$t (translation of $$e).

Items with process statuses are considered “unavailable”. Note that Col.3 of tab_expand can be used to specify item process statuses that their items should be treated as “available”.

To ignore item process statuses that should be treated as “available”, set the following parameter in column 3 of tab_expand: `AVA=BD,MK`. It is possible to specify more than one process statuses, delimited by a comma “,”.

If you want to take reshelving time into account; set the following parameter in column 3 of tab_expand: `RESHELVING=Y`. This is mostly relevant for Real Time Availability functionality (availability X service).

For example:

```
!   1                   2                              3
!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!>
FULLP      expand_doc_bib_avail_hol       THRESHOLD=050;AVA=BD,NW;SF=z,t
```

In this example:
The THRESHOLD parameter limits the number of items per sublibrary to 50.
The AVA parameter defines items with process status BD or NW as available.
The SF parameter adds 852$$z and 852$$t to the AVA$$z and AVA$$t. The SF parameter supports multiple subfield occurrences.

Note:

For items that are not related to any HOL records, an AVA field is created for each sublibrary and collection combination similar to expand_doc_bib_avail, as if `COLLECTION=Y` is defined and without consulting the `SF` parameter.

expand_doc_bib_hol

The `expand_doc_bib_hol` program adds holdings data (the holdings record) to the bibliographic record.

Note that this expand routine's behavior is the same as `expand_doc_hol_bib`.

The program arguments (column 3 in tab_expand) are:

1. MERGE_TYPE=merge no. Note that it is possible for more than one HOL record to be merged into the ADM or BIB record.

2. SUP-HOL=Y/N. To enable the expanding of all holdings fields into the linked bib document even when the holding record has the STA=SUPPRESSED field.

3. Fields such as 856.

expand_doc_bib_hol_ana

the expand_doc_bib_hol_ana program finds an LKR field in the BIB record with subfield 'a' = "ANA", takes the related BIB record number from subfield 'b', and expands its Holdings record information in the same way as expand_doc_bib_hol works.
You can use col. 3 in `tab_expand` for parameter definition, just as in `expand_doc_bib_hol`.

Here is an example:

BIB record 10, has an LKR field, $$aANA$$b000000020.

BIB record 20, has a HOL record number 30.

The result of applying this expand program to BIB record 10 is that fields from HOL 30 will be added to it.

expand_doc_bib_hol_usm

The `expand_doc_bib_hol_usm` program takes the 866 MARC21 field (textual holdings - basic bibliographic unit) of the holdings record and concatenates it with the 852 MARC21 field (location) of the holdings record, creating a new 866 field. In addition to the new 866 field, the program adds the holdings record to the bibliographic record.

Note that if the holdings record has field STA, the record is displayed in the Web OPAC only if the field text is "DISPLAY". If the record does not have an STA field, the record will be displayed.

Note: To control the display of the 852 subfield $z in the 866 field, column 3 of tab_expand should include the parameter SUPPRESS_SF_Z as in the following example:

```
!   1                    2                              3
!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!>
U39-DOC    expand_doc_bib_hol_usm         SUPPRESS_SF_Z
```

expand_doc_bib_hol_usm_2

The `expand_doc_bib_hol_usm_2` program takes the 866 MARC21 field (textual holdings - basic bibliographic unit) of the holdings record and concatenates it with the 852 MARC21 field (location) of the holdings record, creating a new 866 field.

expand_doc_bib_inv

The `expand_doc_bib_inv` program builds an INV field from the inventory number of the item record (Z30-INVENTORY-NUMBER). The structure of the INV field is:

```
INV $a [inventory number]
```

expand_doc_bib_lng_cjk

The `expand_doc_bib_lng_cjk` expand program adds subfield $$9 with the language code (chi, jpn, and kor) to each CJK field, in order to differentiate between Chinese, Japanese and Korean. This subfield facilitates separate indexing for CJK languages. The expand program adds the subfield to all fields that have CJK characters.

expand_doc_bib_loc_1_a

This expand imports the holdings library code (XXX60), the holdings system number and the indicators and subfields of the MARC21 location field (852) into the temporary PS1 field.

expand_doc_bib_loc_1_b

This expand imports items into the temporary PS1 field using links of type ADM. This should be used in the bibliographic library (XXX01) and in Course Reading libraries (XXX30). If the parameter 'HOL-LIBRARY=N' is set for this expand, the PS1 fields are created even if the expanded BIB record is not connected to any HOL library. In this case, the ALEPH string is used instead of the value of the HOL library in the PS1$$r subfield.

expand_doc_bib_loc_1_b2

This expand imports items into the temporary PS1 field using links of type ITM. The expand should be used in Course Reading libraries (XXX30) and in any regular bibliographic library that uses ITM links (for example, analytical records).

Note that for expand_doc_bib_loc_1_b and expand_doc_bib_loc_1_b2 the Z16 is not included as in expand_doc_bib_loc_usm and expand_doc_bib_psts.

expand_doc_bib_loc_1_c

In order to set priorities for processing status over item status, this expand stores the item process in subfield $e. If the item is not in process, this expand routine takes the loan status of the item and stores it in subfield $d.

Creates PST directly from the holdings record, bypassing the creation of the temporary PS1, if there are no items linked to the holdings record.

This program should be used for sites where the items and the holdings records are linked.

expand_doc_bib_loc_1_c2

This program is like the `expand_doc_bib_loc_1_c` program except that it does not create the PST directly from the holdings record if there are no items linked to the holdings record.

This program should be used for sites where the items and the holdings records are not linked.

expand_doc_bib_loc_3_a

This expand program adds the following subfields (replaces codes by names) for display purposes:
$3 - Material type (display form)
$4 - Sublibrary name
$5 - Collection name
$6 - Item loan status (display form)
$7 - Item process status (display form)

expand_doc_bib_loc_4_a

Imitates `expand_doc_bib_loc_usm` creating a LOC field.

expand_doc_bib_loc_4_b

Imitates `expand_doc_bib_psts` creating a PSTS field.

expand_doc_bib_loc_4_c

Imitates `expand_doc_bib_loc_usm` creating the SBL, LOC and STS fields for linked item and holdings records.

expand_doc_bib_loc_5_c

Imitates `expand_doc_bib_loc_usm` creating the SBL, LOC and STS fields for linked subscription records.

expand_doc_bib_loc_cleanup

This program removes the intermediate PS1 fields.

expand_doc_bib_loc_dedup

The expand_doc_bib_loc_dedup program prevents duplicate locations for serials with subscriptions containing the same call number as the items. To prevent duplicate locations, a line with expand_doc_bib_loc_dedup must be added to tab_expand, after the line with expand_doc_bib_loc_usm.

expand_doc_bib_loc_disp

The `expand_doc_bib_loc_disp` program expands subfields $b, $c and $o of the LOC field created by `expand_doc_bib_loc_usm`, adding subfields $4 (sublibrary), $5 (collection) and $3 (material type) in which the codes are replaced by names.

expand_doc_bib_loc_n and expand_doc_sort_loc_x

The following expand programs are used to include location information in bibliographic indexes and displays. They are a modular set of expand programs that integrate the functionality of expand_doc_bib_loc_usm and expand_doc_bib_psts.

This expand mechanism generates intermediate PS1 fields; the PS1's are sorted and deduplicated into PST fields. Codes (for example, sublibrary) in the PST fields are expanded into display forms.

Structure of the PST field:

1st indicator: call number type (0-8).
2nd indicator: undefined, contains a blank.
$$0 [origin of the PST field].
If the field originates from an item record, then the subfield contains Z30 ($$0Z30).
If the field originates from a holdings record, then the subfield contains HOL ($$0HOL).
$$1 [unique identifier of the record of origin]
If the field originates from an item record, then the subfield contains the system number of the linked administrative record and the item sequence number (for example, $$11000005921000010). Format: <Z30-DOC-NUMBER> <Z30-ITEM-SEQUENCE>
If the field originates from a holdings record, then the subfield contains the holdings library code and the system number of the holdings record (for example, $$1USM60-000001909). Format: <library code>60;holdings system number>
$$b [sublibrary code].
$$c [collection code].
$$d [item status] if there is no item process status.
$$e [item process status] if there is an item process status in the item record.
$$f [temporary location flag].
If the sublibrary, collection and call number information are temporary (the Temporary Location box is checked), then the subfield contains Y ($$fY).
If the location is not temporary, then the subfield contains N ($$fN).
$$h [call number] if call number type is 0-3 or 6-8.
$$j [call number] if call number type is 4.
$$l [call number] if call number type is 5.
$$n [call number type]
$$o [material type - column 1 of the tab25.lng table] (for example, BOOK). Any item material type with first three letters ISS, such as ISSBD, will get $$0ISSUE.
$$r [linked holdings record]
Contains the holdings library code and the system number of the holdings record linked to the item (for example, $$1USM60-000001909). Format: <library code>-<holdings system number>
$$y [copy number]
$$3 [material type - display form: column 3 of the tab25.lng table] (for example,

Book).
$$4 [sublibrary name]
$$5 [collection name]
$$6 [item loan status - display form]
$$7 [item process status - display form]

The PST field is only created from the linked item and holdings records and not for the linked subscription records.

Following is the list of the programs:

expand_doc_bib_loc_1_a

expand_doc_bib_loc_1_b

expand_doc_bib_loc_1_b2

expand_doc_bib_loc_1_c

expand_doc_bib_loc_1_c2

expand_doc_sort_loc_a

expand_doc_sort_loc_b

expand_doc_bib_loc_3_a

expand_doc_bib_loc_4_a

expand_doc_bib_loc_4_b

expand_doc_bib_loc_4_c

expand_doc_bib_loc_5_c

expand_doc_bib_loc_cleanup

expand_doc_hol_loc_1_a

expand_doc_hol_loc_2_a

In addition, note that a Z07 record is triggered for the bibliographic record linked to the item when one of the following fields of the item record is updated:

Z30-BARCODE

Z30-SUB-LIBRARY

Z30-MATERIAL

Z30-ITEM-STATUS

Z30-COLLECTION

Z30-CALL-NO-TYPE

Z30-CALL-NO

Z30-CALL-NO-KEY

Z30-CALL-NO-2-TYPE

Z30-CALL-NO-2

Z30-CALL-NO-2-KEY

Z30-DESCRIPTION

Z30-INVENTORY-NUMBER

This ensures that the expanded bibliographic record is updated when information related to the item is changed.

The following is an example of the setup for a site where items are linked to holdings records:

```
!   1                       2
!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
XXX-XXX    expand_doc_bib_loc_1_a
XXX-XXX    expand_doc_bib_loc_1_b
XXX-XXX    expand_doc_bib_loc_1_b2
XXX-XXX    expand_doc_bib_loc_1_c
XXX-XXX    expand_doc_sort_loc_b
XXX-XXX    expand_doc_bib_loc_2_a
XXX-XXX    expand_doc_bib_loc_3_a
XXX-XXX    expand_doc_bib_loc_cleanup
```

Following is an example of the setup for a site where items are not linked to holdings records:

```
!   1                       2
!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
XXX-XXX    expand_doc_bib_loc_1_a
XXX-XXX    expand_doc_bib_loc_1_b
XXX-XXX    expand_doc_bib_loc_1_b2
XXX-XXX    expand_doc_bib_loc_1_c2
XXX-XXX    expand_doc_sort_loc_a
XXX-XXX    expand_doc_bib_loc_2_a
XXX-XXX    expand_doc_bib_loc_3_a
XXX-XXX    expand_doc_bib_loc_cleanup
```

expand_doc_bib_loc_usm

The `expand_doc_bib_loc_usm` program builds four fields from the Z30 (item record), the Z16 (subscription record), and the 852 field (location) of the holdings record: SBL, LOC, STS and PST.

Another field, HLD, is created based on PST fields with $$0HOL. It contains a link to the relevant items, Electronic Location, Summary Holdings Information, Index Holdings and Supplement Holdings. This field can be presented ONLY in the full view screen. It will not work in brief view.

There is a program argument, OPTIMIZE=N, which, when added to the 3rd column of `tab_expand`, causes all items to be expanded, including those with different call number information. When OPTIMIZE=N is used, the run time is not optimized.

Here is an example:

```
!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!>
WORD       expand_doc_bib_loc_usm         OPTIMIZE=N
```

In Course Reading/Reserves, expand_doc_bib_loc_usm works only for records whose xxx30 doc is linked to the item via a (z103) ADM-type link; it does not work for ITM-type links:
- the xxx30 professors' copies which are linked to the xxx50 item directly have a z103_lkr_type ADM;
- the xxx30 records which are linked to the xxx50 item via an xxx01 bib record have a z103_lkr_type ITM.

The expand_doc_bib_loc_usm is a hard-coded set of calls to expand_doc_bib_loc_n_x programs.
expand_doc_bib_loc_1_b reads items linked via an ADM-type link;
expand_doc_bib_loc_1_b2 reads items linked via an ITM-type link.
expand_doc_bib_loc_1_b2 is not one of the programs called by expand_doc_bib_loc_usm.

In order to include both LKR types (ADM and ITM), the xxx30 tab_expand needs to specify both _b and _b2:

```
CREATE-Z13 expand_doc_course
CREATE-Z13 expand_doc_bib_loc_1_a
CREATE-Z13 expand_doc_bib_loc_1_b
CREATE-Z13 expand_doc_bib_loc_1_b2
CREATE-Z13 expand_doc_bib_loc_1_c
CREATE-Z13 expand_doc_sort_loc_b
CREATE-Z13 expand_doc_bib_loc_2_a
CREATE-Z13 expand_doc_bib_loc_3_a
CREATE-Z13 expand_doc_bib_loc_4_a
```

Structure of the SBL field:

Indicators - both undefined, each contains a blank.
$a [sublibrary code]

Structure of the LOC field:

1st indicator: Call number type (0-8).
2nd indicator: undefined, contains a blank.
$b [sublibrary code]
$c [collection code]
$h [call number] if call number type is 0-3 or 6-8.
$j [call number] if call number type is 4.
$l [call number] if call number type is 5.
$o [material type]

Structure of the STS field:

Indicators - both undefined, each contains a blank.
$a [item status code]

This program uses the same environment variable that is used when ALEPH automatically updates the Z16 (subscription record) and the Z30 (item record) from the 852 field of the linked holdings record. The program only expands the subfields of the 852 field defined in the correct_852_subfields environment variable defined in the `aleph_start` file. In this way, call numbers from the item and the holdings record are treated consistently when they are merged into a single list during the expand.

Additionally, note that a Z07 record is triggered for the bibliographic record linked to the item when one of the following fields of the item record is updated:

- Z30-SUB-LIBRARY
- Z30-MATERIAL
- Z30-ITEM-STATUS
- Z30-COLLECTION
- Z30-CALL-NO-TYPE
- Z30-CALL-NO
- Z30-CALL-NO-KEY
- Z30-CALL-NO-2-TYPE
- Z30-CALL-NO-2
- Z30-CALL-NO-2-KEY
- Z30-DESCRIPTION
- Z30-INVENTORY-NUMBER

A Z07 for the bibliographic record is also triggered when the linked subscription record is updated.

This ensures that the expanded bibliographic record is updated when information related to the linked item or to the linked subscription information is changed.

Structure of the PST field:

See above.

expand_doc_bib_local_notes
This program is used to expand into the bibliographic record - for display and indexing purposes - local tags stored in the holdings record. The application for storing local tags in a holdings record is in a consortial environment where a single bibliographic record is shared by multiple institutions and an institution would like to include local tags that not everyone can see. Which local tags are moved to the holdings record from the bibliographic record (see the entry for `fix_doc_create_hol_local_notes`) and which local tags from the holdings records are displayed in the OPAC can be configured by the local institution.

When the records are displayed in their full format (WEB-FULL routine from `tab_expand`) or indexed, the program consults the `tab_base.conf` table of the `alephe` directory, and only the local tags of the selected base are indexed and displayed.

The following are the tables involved:

The `tab_expand` in the tab directory of the bibliographic library (XXX01):

Determines the instances in which the program is performed. The following is a sample of the setup needed for displaying the local tags from the holdings record together with the bibliographic record:

```
WEB-FULL   expand_doc_bib_local_notes
```

The tab_expand_local_notes.conf of the bibliographic library (XXX01):

This table is used to define the owner codes for the holdings records to display local tags in the OPAC. The table also defines the tags that are to be expanded. In the sample below, XXX01_AA will display the local tags that are in the holdings records where the OWN tag is AA. XXX01_PUB will display local tags for both AA and BB owners:

```
[XXX01_AA]
owners list = AA
owner tag = OWN
owner subfield = a
owner alternative tag = 590,690
owner alternative subfield = 9
mapping section = LCN-2-BIB

[XXX01_PUB]
owners list = AA,BB
owner tag = OWN
owner subfield = a
owner alternative tag = 590,690
owner alternative subfield = 9
mapping section = LCN-2-BIB
```

The `tab_mapping` table in the tab directory of the bibliographic library (XXX01):

This table is used to map the local tags into the new virtual tags to be indexed or displayed. In the sample below, if the holdings record has a 690 field, then the program expands this field into the virtual LCS field in the bibliographic record:

```
LCN-2-BIB       541## abcde LCN   abcde                                Y Y
LCN-2-BIB       541## fho39 LCN   fho39                                Y Y
LCN-2-BIB       561## ab39  LCN   ab39                                 Y Y
LCN-2-BIB       590## ab9   LCN   ab9                                  Y Y
LCN-2-BIB       690## ab9   LCS   ab9                                  Y Y
```

Note that for the indexing and/or display of the new virtual fields, it is necessary to further customize the standard display and indexing tables (for example, edit_doc_999.lng, tab11_word).

expand_doc_bib_multi_lng

For multilingual applications, the `expand_doc_bib_multi_lng` program adds other language fields to the record. The program adds all the headings (Z01 records) that are linked to the same authority record as the heading field and that are not cross-references.

Note that `expand_doc_bib_accref` includes `expand_doc_bib_multi_lng`. Therefore there is no need to have it listed under the WORD system function. The `expand_doc_bib_multi_lng` program must only be used with the ACC system function.

In order to populate subfield $$6 in the BIB record after its enrichment, set column 3 of the tab_expand table with the parameter `add_sf6=Y`.

expand_doc_bib_ndu
The `expand_doc_bib_ndu` program creates a virtual TIT5 field for the following MARC 21 fields:

130, 245, 730 (from all subfields)
240, 242, 243, 246, 770 (from subfield $a)
700, 710, 711, 773, 780, 785 (from subfield $t)

The program strips punctuation, capitalizes text and removes initial articles (if suppressed for filing).

expand_doc_bib_psts
The `expand_doc_bib_psts` program builds a PSTS field from the Z30 (item record), the Z16 (subscription record), and the 852 field of the holdings record. This routine shows the processing status of the item record if available, as well as the call number, collection and sublibrary.

Structure of the PSTS field:

Indicators - both undefined, each contains a blank:
$b [sublibrary code]
$c [collection code]
$h [call number] if call number type is 0-3 or 6-8.
$j [call number] if call number type is 4.
$l [call number] if call number type is 5.
$d [item status] if no item process status.
$e [item process status] if there is an item process status.

Note that the item process status is stored in subfield $e. If the item is not in process, the expand routine takes the loan status of the item and stores it in subfield $d.

This program uses the same environment variable that is used when ALEPH automatically updates the Z16 (subscription record) and the Z30 (item record) from the 852 field of the linked holdings record. The program only expands the subfields of the 852 field defined in the `correct_852_subfields` environment variable defined in the `aleph_start` file. In this way, call numbers from the item and the holdings record are treated consistently when they are merged into a single list during the expand.

Note that the `expand_doc_bib_psts` program extracts items linked to the bibliographic record through ADM and ITM links. It is necessary to extract ITM links so that the expand program will display items in a Course Reading library whether or not the ADM record is linked to the Course Reading document or to a bibliographic document.

This expand routine skips those holdings records that have been suppressed (STA$$aSUPPRESSED).

Additionally, note that a Z07 record is triggered for the bibliographic record linked to

the item when one of the following fields of the item record is updated:

- Z30-SUB-LIBRARY
- Z30-MATERIAL
- Z30-ITEM-STATUS
- Z30-COLLECTION
- Z30-CALL-NO-TYPE
- Z30-CALL-NO
- Z30-CALL-NO-KEY
- Z30-CALL-NO-2-TYPE
- Z30-CALL-NO-2
- Z30-CALL-NO-2-KEY
- Z30-DESCRIPTION
- Z30-INVENTORY-NUMBER

A Z07 for the bibliographic record is also triggered when the linked subscription record is updated.

This ensures that the expanded bibliographic record is updated when information related to the linked item or to the linked subscription information is changed.

expand_doc_bib_psts_disp
The `expand_doc_bib_psts_disp` program expands subfields $b and $c of the PSTS field created by `expand_doc_bib_psts`, adding subfields $4 (sublibrary name) and $5 (collection name) in which the codes are replaced by names.

Note that `expand_doc_bib_psts` is intended for indexing, while `expand_doc_bib_psts_disp` is intended for display.

expand_doc_bib_subtype

This expand routine retrieves a list of subject fields as a parameter and adds the subject type to each of them, as follows:

- If the second indicator is 4: The source is not specified. No changes are done.
- If the second indicator is 7: The source is already specified in subfield 2. No changes are done.
- If the second indicator is neither 4 nor 7, subfield 2 with the value of the second indication is added to the subject fields.
- If the field already has a value in subfield 2, it is removed (unless the second indication is 7 or 4).

expand_doc_bib_tab04
The `expand_doc_bib_tab04` program is primarily intended for the Z39_SERVER instance in the `tab_expand` table of the library's tab directory. This program can be used to translate alphabetic tags into numeric values (for example, LOC to 952). Note that the Z39 protocol does not recognize non-numeric tags and ALEPH fields (such as LOC, CAT, Z30, and so on) need to be converted using the `expand_doc_bib_tab04` program.

The program works with the `tab04` table of the library's `tab` directory. The `tab04` table is used to set up the specification for the conversion of one set of cataloging tags

to another. The expand_doc_bib_tab04 program uses entries defined under the conversion routine 90. Following is a sample from the tab04 table:

```
90 Z####            9#### N
90 CAT##            956   N
90 #####            ##### N
```

Note that the last line in this sample should always be present.

expand_doc_bib_z30

This program is used to expand the item's information into the bibliographic record. The expand_doc_bib_z30 program is used with the expand_doc_bib_z30 table of the library's tab directory. The expand program creates a new virtual field - Z30-1 (for copy items) or Z30-2 (for issue items) - that contains the item's information. The table is used to define which fields from the item record are expanded and to determine the subfield structure of the new expanded field.

The parameters column of the tab_expand table can be used in order to determine whether to consult the tab15.lng table, which table used be used to determine the structure of the new field and which field should be created. The following parameters are available:

- CONF - This option can be used in order to determine that a different table should be used instead of the default expand_doc_bib_z30 table. Following is a sample of the usage of the variable:

  ```
  !   1                   2                          3
  !!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-!!!!!!!!!!!!!!->
  U39-DOC    expand_doc_bib_z30             CONF=exl_z30
  ```

  If CONF is not set, the default expand_doc_bib_z30 table is used.

- TAB15 - This parameter can be used to specify whether or not column 10 of the tab15.lng table should be consulted. This column is used to specify whether the copy should be displayed in the Web OPAC. If the TAB15 variable is set to Y in the tab_expand table, then if column 10 of the tab15.lng table contains an 'N' for the type of items attached to the bibliographic record, the expand does not create new virtual fields for these items. The following options are available in order to specify that virtual fields should be created disregarding the specifications of the tab15.lng table:

  o Set TAB15 to N (TAB15=N) or it is not defined.
  o Column 3 of the tab_expand table contains the string ALL.

  Following is a sample line for the usage of this parameter:

  ```
  !   1                   2                          3
  !!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-!!!!!!!!!!!!!!->
  WEB-FULL   expand_doc_bib_z30             TAB15=Y
  ```

- TAG - This parameter can be used in order to specify that a different new expanded field should be created instead of the defaults Z30-1 (for copy items) or Z30-2 (for issue items).

  Following is a sample line for the usage of this parameter:

  ```
  !   1                   2                          3
  !!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-!!!!!!!!!!!!!!->
  ```

```
U39-DOC     expand_doc_bib_z30              TAG=LOC
```
If this parameter is not set, Z30-1 and Z30-2 will be used.

Note that this program should be used with special care as it can create performance problems, due to the potential overflow of the buffers. This program should be avoided by libraries that have numerous item records linked to one bibliographic record (usually due to the individual listing of serial issues).

expand_doc_bib_z403 (functional for ADAM)
The expand_doc_bib_z403 program can be used to expand the object's data information into the bibliographic record. The expand_doc_bib_z403 program is used with the expand_doc_bib_z403 table of the library's tab directory. The expand program creates a new virtual field, Z403, that contains the object's information. The table also determines the subfield structure of the new expanded field.

The following parameters can be specified in the tab_expand table in order to modify the program's defaults:

- TAG - This parameter can be used in order to specify that a different new expanded field should be created instead of the default Z403 field.
- USAGE-TYPE - This parameter can be used in order to specify the type of objects that are included in the expand window. By default, the program only expands information from objects where the Z403-USAGE-TYPE is set to VIEW. This parameter can be used to specify that other object types should be included. In order to include all types of Z403-USAGE-TYPE, the parameter can be set to ALL. In order to include more than one type but not all types you can either repeat the line in the tab_expand table or repeat the parameter in the same line (for example, USAGE-TYPE=INDEX,USAGE-TYPE=VIEW).
- CONF - This option can be used in order to determine that a different table should be used instead of the default expand_doc_bib_z403 table.
- DISPLAY-LINK - If this parameter is set to "N", the object's data is not expanded when the expiry date of the object has been reached or/and when the Show in OPAC flag is set to No.

expand_doc_bnu_initials
This expand program adds a virtual subfield $G to UNIMARC fields 701, 702, 711, 712 and 200. The virtual subfield is built from the pinyin translation of these fields which is stored in subfield $A. The subfield contains the initials of the contents of subfield $A (initials of the intellectual responsibility and/or title). Note that this expand program is to be used by Chinese installations only and enables the retrieval of records using initials.

expand_doc_course
The `expand_doc_course` program must be used for the implementation of Course Reading Management. The program should be present in the tab_expand table of the Course Reading library (XXX30) under all instances (system function).

expand_doc_crs_bib
The expand_doc_crs_bib is useful when the BIB record is expanded from another BIB record and the course record should have information from both the related BIB record and its expanded document. The program can be present in the tab_expand table of the Course Reading library (XXX30) under all instances (system function).

When setting expand_doc_crs_bib, define in col. 3 of tab_expand the fields that will be expanded to the course document.

For example: ./xxx30/tab/tab_expand

```
CREATE-Z13                  expand_doc_crs_bib              331-2,1##-2
```

expand_doc_date_yrr

The expand_doc_date_yrr expand routine facilitates indexing date ranges that are related to the BIB record. The expand creates YRR fields for the range of dates that are indicated in the Publication Date 1 and Publication Date 2 of USMARC records (positions 06,07-10 and 11-14 of field 008) and UNIMARC records (positions 08, 09-12 and 13-16 of subfield $$a of the 100 field).

expand_doc_del_fields

This expand program deletes all the fields in the record except the fields specified in Col.3 of tab_expand. The fields in Col.3 are separated by a comma.

Example:

```
!   1                   2                              3
!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!>
TEST       expand_doc_del_fields          245##,260##,500##
```

Only 245##, 260##, 500## and AVA## fields will be retained.

expand_doc_deleted

This routine deletes all the expanded lines of a deleted record and should be added at the end of the sections relevant for indexing.

expand_doc_duplicate_field

The expand_doc_duplicate_field program is used to duplicate a field, assigning a new field tag plus indicators. The expand_doc_duplicate_field program is used with the tab_expand_duplicate_field table of the library's tab directory. This table defines the field to be duplicated and the new assigned field tag. For example, if the tab_expand_duplicate field contains the following line:

```
! 1   2
!!!!!-!!!!!
260## IMP
```

then the 260 field is duplicated and assigned to a new IMP field. The expand_doc_duplicate_field procedure and expand_doc_duplicate table can be used to overcome the problem created when using expand_doc_split, which does not retain the source field. When expand_doc_split is based on a field created by the expand_doc_duplicate_field program, the source field is retained. In the above example, the IMP field can be later processed by expand_doc_split without losing the original 260 field.

expand_doc_extract

The expand_doc_extract program is used with the tab_expand_extract table of the library's tab directory. This table defines extraction of subfields for indexing. For example, if the library wants to create a headings list of "chronological subdivisions" for "subject added entries - topical terms", it is possible to define that MARC21 650 field, subfield $y is to be expanded into a new tag (for example, y650). The virtual field may then be indexed or displayed.

Note that the fourth column in the tab_expand_extract table can be used to specify the number of subfield occurrences for which the new virtual field is created. For example, you can define that only the first occurrence of subfield $y in the 650 field should be used for the creation of the new field. The following is an extract from the table:

```
! 1   2  3    4
!!!!!-!-!!!!!-!
650## y Y650  1
```

expand_doc_extract_holding

This program moves the 852 field from the holdings record into the linked bibliographic record. In addition, the contents of subfields $a and $z of the 866/7/8 fields of the holdings records are added to the new 852 field under subfield $3. Note that between each additional subfield and field, three asterisks are inserted (***). Subfield $a of the new field is built based on the library code (first three positions, for example USM). For example, if the holdings record contains the following fields:

```
86631 L $$aav. 37-52$$zSome issues missing
```

a new 852 field is added to the linked bibliographic record as follows:

```
8520  L $$aUSM$$bUELEC$$hhE183.8.B7$$iL494$&
#36;3av. 37-52***Some issues missing
```

The parameters column of the tab_expand table can include two program arguments: the first calls a section in tab_fix of the holdings library (XXX60) and the second lists the item processing status that should be considered as suppressed (the expand should skip the holdings records linked to the items). Note that in order to use this expand routine, a line containing ADM USM60 USM50 must be present in the library_relation table. For example, if tab_expand is defined as follows:

```
!   1                   2                              3
!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!>
U39-DOC    expand_doc_extract_holding     HOL,OI,BP,CA,CL,CT
```

then HOL is a section from the tab_fix table of the holdings library and OI, BP, CA, CL and CT are suppressed item processing statuses. Up to 10 item process statuses can be defined.

In addition, the program does not expand holdings records where the STA field is set to "SUPPRESSED". Additionally, the program does not check records in SE format.

Note that the hash (#) character can be used as a wildcard and that you can specify the list to have values such as A# or #B. This will filter out all process status codes starting with A or ending in B.

expand_doc_fix_abbreviation

The expand_doc_fix_abbreviation program is used to change abbreviations into full text. The routine can be used to replace any text string in a record with a different text string. There are two options:

A new duplicate field is added to the record with the non-abbreviated form of the text.

The abbreviated form of the text is changed into full text in the original field.

In order to specify whether a new duplicate field is added to the record with the non-abbreviated form of the text or whether the abbreviated form of the text is replaced in the original field with the new form, use the parameters column in `tab_expand` (col. 3).

The available values are ADD and REPLACE. ADD adds a new field to the record with the non-abbreviated form and REPLACE replaces the abbreviated form with the new form in the original field without duplication. If the column is left blank, the system duplicates the field (as ADD). Following is a sample of the `tab_expand` table:

```
!   1                   2                              3
!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!>
U39-DOC    expand_doc_fix_abbreviation    ADD
```

The `expand_doc_fix_abbreviation` program is used with the `tab_abbrev` table of the library's `tab` directory (see `tab_abbrev`). The `tab_abbrev` table contains the list of "abbreviations" and the whole forms into which the shortened form is to be changed in the new virtual field added to the record.

Following is a sample of the `tab_abbrev` table:

```
! 1   2         3                              4
!!!!!-!-!!!!!!!!!!!!!!!!!!!!-
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
225## Y 1st            FIRST
043## Y u-at---        Australia
043## Y u-at-ne        Australia New South Wales
```

Based on the above sample, the following document:

```
043   L $$au-at-ne

1001  L $$aWilliams, David

2451  L $$aThe 1st man
```

is expanded into:

```
043   L $$au-at-ne

1001  L $$aWilliams, David

2451  L $$aThe 1st man

043   L $$aAustralia New South Wales

2451  L $$aThe FIRST man
```

Note that this program also can be used as a fix program (using `tab_fix` in place of `tab_expand`). The difference is that instead of adding virtual fields (for indexing or/and display), when used as a fix program, the fields with the whole forms are actually added to the record.

expand_doc_fmt

The `expand_doc_fmt` program builds a TYP field from the record's format (FMT field).

Structure of the TYP field:

Indicators - both undefined, each contains a blank:
$a [record's format code]
$b [record's format name]

expand_doc_fmt_mgu
The `expand_doc_fmt_mgu` program builds a TYP field indicating the type of record. The TYP field is created based on coding in the LDR (positions 06 and 07), 006 (position 00), 007 (positions 00 and 01), and 008 (position 23) fields.

Structure of the TYP field:

Indicators - both undefined, each contains a blank.
$a [<type of record>]

The TYP field is generated with one of the following:

```
<Electronic Resource>
<Web Resource>
<Microform>
<Serial>
<Electronic Journal>
<Web Journal>
<Microform Serial>
<Computer File>
<Map>
<Digital Map>
<Score>
<Sound>
<Archive/MSS>
<Visual>
<Graphic>
<Kit>
<Realia>
```

expand_doc_hld_stmt

The `expand_doc_hld_stmt` routine generates 863/4/5 enumeration and chronology data in HOL records, using item records that are linked to the HOL record. The fields that are generated can then be used to create compressed holdings statements for display.

As an expand routine, the compressed holdings statements are built on-the-fly when the record is displayed. Note that a similar program, fix_doc_hld_stmt, can be applied in the cataloging module (after setting the tab_fix and the fix_doc.eng table of the HOL library (XXX60). When the fix_doc_hld_stmt program is performed, the 863/4/5 fields are added to the HOL record.

This routine is relevant to HOL libraries only. Note that this routine is relevant only if the 85x/85xX Publication Pattern fields reside in the HOL document record.

The following item records are taken into consideration when generating the 863/4/5 fields:

Items that have HOL no. in the Hol. Link field.

Items for which the enumeration field is equal to or greater than the value entered in subfield $$a of S63/4/5 (Starting Point field) in the HOL record. If there is no S63/4/5 field, enumeration in the item record is not checked. See paragraph 5 for more details.

Item groups:
All items which share the same Linking Number identifier in $$8 and the same Copy-ID (if there is one) are taken into consideration for a single holdings statement; if one HOL record has more than one linked Copy-ID, each Copy-ID creates a separate holdings statement. See paragraph 5 for more details.

"Starting Point" and "Copy-ID":
A special field, S63/4/5 in the HOL record, defines the item's Starting Point and the Copy-ID for generating a holdings statement. This field should be used by libraries that already have holdings statement fields for item records. Use field S63/4/5 with subfield $$a for the volume's Starting Point and subfield $$t for Copy-ID.

For example: if the field "S63" contains in subfield $a the value "3", only items with the first enumeration level of "3" (or higher) will be included in the holdings statement.

If the field "S63" subfield $t contains the value "2", only items with the value "2" in the copy number field (Z30-COPY-ID) will be included in the generated holdings statement.

```
S63   L  $$a3 $$t2

C6340 L $$81.1$$a3$$b2-4$$i2002$$j03-07$$wg$$t2$$9Y
C6340 L $$81.2$$a4$$b1-2$$i2003$$j01-03$$wg$$t2$$9Y
C6340 L $$81.3$$a4$$b4-5$$i2003$$j07-09$$wg$$t2$$9Y
```

Note: it is possible to have several "S63" fields in the HOL record. In this case, a set of holdings statement (863/4/5) fields will be created for each "S63" field that includes subfield $t (copy-ID).

For example:

```
86340 L $$81.13$$a1$$i20000-2000$$t1$$9Y
86340 L $$81.14$$a2$$b1-3$$i2001$$j01-05$$wg$$t1$$9Y
86340 L $$81.15$$a2$$b5-6$$i2001$$j09-11$$t1$$9Y

86340 L $$82.13$$a1$$i2000$$t2$$ $$9Y
86340 L $$82.14$$a2$$b1$$i2001$$j01$$wg$$t2$$9Y
86340 L $$82.15$$a2$$b3$$i2001$$j05$$wg$$t2$$9Y
```

The following procedures are performed by the routine:

Holdings statements (863/4/5 fields) which have $$9 with the value "Y" are deleted.

(Note: $$9Y is used to denote that the holdings statement field should be regenerated each time the procedure is used. When a holdings statement field (863/4/5) is generated, it always includes $$9Y. If the library has holdings statement fields that it wants to retain when running the procedure, the fields should contain $$9N.

Sets of "considered" item records (as defined in section 4 above) are compressed to create ranges by enumeration and chronology.

For example:

```
86340 L $$82.1$$a1$$b1-6$$i2000$$j01-06$$t00001$$9Y
86340 L $$82.2$$a2$$i2001$$t00001$$9Y
```

Each range is written in a separate 863/4/5 field, with the correct break indicator in $$w, if needed. Subfield $$w with the value "g" indicates a gap break (i.e., there are issues that did not arrive). Subfield $$w with the value "n" indicates a non-gap break (i.e., there are issues that were not published).

For example:

```
C6340 L $$82.3$$a3$$b2-5$$i2002$$j03-09$$wg$$t1
```

In the above example, subfield $$w contains the value "g" to indicate that there is a gap break. Issues no. 1 and no. 6 of Volume no. 3 did not arrive.

Break indicator ($$w) is generated for items that are not included in the summary holdings statement. These are Items with the process status of: NA or NP, or items that were mapped to be considered as if they have processing status NA or NP, using the `tab_hld_stmt` table of the administrative library (XXX50).

Since different institutions might use different codes to describe the process-status Not Arrived or Not Published, `tab_hld_stmt` is used to map those different codes into "NA" or "NP".

By using this table, we can also map items that have a particular sublibrary and/or collection, and/or item status, and/or item process status, and/or break indicator to be considered as NA or NP. Those items will be excluded from the holdings statement.

For example, a library that might want to exclude missing items from the holdings statement should map the process status MI (missing) as NA in `tab_hld_stmt`, as follows:

```
! 1     2   3  4  5 6
!!!!!-!!!!!-!!-!!-!-!!
##### ##### ## MI # NA
```

`expand_doc_hld_stmt` should be listed in col.2 of `tab_expand` of the HOL library; for example:

```
!   1                    2                    3
!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!-!!!!!!!!!!!!!!!!>
WEB-FULL   expand_doc_hld_stmt
WEB-FULL   expand_doc_hol_86x
```

The routine will create 863/4/5 tags in UTIL F/4/doc and in WEB FULL display.

Note: It is possible to define alternative field tags instead of 863/4/5 fields in order to differentiate between 863/4/5 fields already present in the record, and field tags generated by the routine.

For example, if `tab_expand` is defined as follows:

```
!   1                    2                    3
```

```
!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!-!!!!!!!!!!!!!!!!!!>
U39-DOC    expand_doc_hld_stmt    CMPRS_TAG=C63
```

The holdings statement field tag will be C63:

```
C6340 L $$81.1$$a1$$i2000$$t2$$9Y
C6340 L $$81.2$$a2$$b1-5$$i2001$$j01-09$$wg$$t2$$9Y
C6340 L $$81.3$$a3$$b2-4$$i2002$$j03-07$$wg$$t2$$9Y
```

This feature can be used if and when you are checking the generated data and want to separate it from data that has actually been written in the record.

To enable the generation of summary holdings statements for item records that are not linked to a subscription record or do not hold a value in the Copy ID field in the subscription record, the parameter:

```
GET_Z30_BY=HOL
```

should be defined in col.3 of `tab_expand` as follows:

```
!   1                    2                              3
!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!>
WEB-FULL   expand_doc_hld_stmt            GET_Z30_BY=HOL
```

For item records that are linked to a subscription record and which hold a value in the Copy ID field in the subscription record and the item records, the parameter:

```
GET_Z30_BY=COPY_ID
```

can be defined in col.3 of `tab_expand` as follows:

```
!   1                    2                              3
!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!>
WEB-FULL   expand_doc_hld_stmt            GET_Z30_BY=COPY_ID
```

Note: If no parameter is defined in col.3 of `tab_expand` the system will use the default parameter:

```
GET_Z30_BY=COPY ID
```

expand_doc_hol_852_disp

The `expand_doc_hol_852_disp` program expands subfields $b and $c of the 852 MARC21 location field adding subfields $4 and $5 in which the sublibrary code and collection code are replaced by names.

expand_doc_hol_86x

The `expand_doc_hol_86x` program is used to create the textual holdings statement fields (866, 867 and 868) based on the 853-855 (Captions and Pattern) and 863-865 (Enumeration and Chronology) fields from the holdings record.

The textual holdings statement fields are created according to the ANSI/NISO Z39.71 - 2006 standard (Holdings Statements for Bibliographic Items) with stripped punctuation.

`expand_doc_hol_86x` creates the text holding statement so that enumeration and chronology data are displayed separately with or without parentheses. This form of display is called `Separate`. The following is an example of a separate display:

```
vol.10:no.2-12(2003:Feb.-Dec.)
```

If the holdings record contains the following fields:

```
85320 L $$81$$av.$$bno.$$u9$$vr$$i(year)$$j(month)$$wm
86332 L $$81.1$$a1-4$$i1941-1944
86332 L $$81.2$$a6-86$$i1945-1987
```

then the textual holdings statement fields are created as follows:

```
86631 L $$av.1-4(1941-1944)
86631 L $$av.6-86(1945-1987)
```

If the holdings record includes an 866/7/8 field with the same sequence number in subfield $8 as the sequence number registered in subfield $8 of the 863/4/5 field, then no other 866/7/8 field is generated by the program.

If the holdings record includes an 866/7/8 field with sequence number 0 registered in subfield $8, then no other 866/7/8 field will be generated by the program.

Note that this expand program must be defined in the `tab_expand` table of the holdings library (XXX60).

In order to update the 866/7/8 fields of the HOL record when item arrival is registered, it is recommended to set the following line in tab_z105 of the ADM library. For example: ./usm50/tab/tab_z105

```
  1            2   3     4     5     6     7     8     9    10     11
12
!!!!!!!!!!!!!!!-!-!!!!!-!!!!!-!!!!!-!!!!!-!!!!!-!!!!!-!!!!!-!!!!!-
!!!!!-!!!!!
UPDATE-ENUM     n USM60
```

expand_doc_hol_86x_iso

This expand routine is the same as `expand_doc_hol_86x_`, except that `expand_doc_hol_86x_iso` creates the textual holding statement so that enumeration and chronology data are displayed adjacent to each other, with enumeration recorded first. This form of display is called `Adjacent`. The corresponding chronology is enclosed in parentheses. The following is an example of adjacent display:

```
vol.10:no.2 (2003:Feb) –vol.10:no.12(2003:Dec.),
```

If the holdings record contains the following fields:

```
85320 L $$81$$av.$$bno.$$u9$$vr$$i(year)$$j(month)$$wm
86332 L $$81.1$$a1-4$$i1941-1944
86332 L $$81.2$$a6-86$$i1945-1987
```

then the textual holdings statement fields are created as follows:

```
86631 L $$av.1(1941)-v.4(1944)
86631 L $$av.6(1945)-v.86(1987)
```

The textual holdings statement fields are created according to the ANSI/NISO Z39.71-2006 standard (Holdings Statements for Bibliographic Items) with stripped punctuation.

Note that this expand program must be defined in the tab_expand table of the holdings library (XXX60).

In order to update the 866/7/8 fields of the HOL record when item arrival is registered, it is recommended to set the following line in tab_z105 of the ADM library. For example: ./usm50/tab/tab_z105.

```
!  1             2    3      4      5      6      7      8      9     10
11    12
!!!!!!!!!!!!!!!-!-!!!!!-!!!!!-!!!!!-!!!!!-!!!!!-!!!!!-!!!!!-!!!!!-
!!!!!-!!!!!
UPDATE-ENUM     o USM60
```

expand_doc_hol_bib

The expand_doc_hol_bib program adds bibliographic data to the holdings record.

The following tables define how the holdings record is updated/merged:

tab_expand in the tab directory of the holding library (XXX60):

The parameters column (Col. 3) specifies the routine name to be applied

Column 3 syntax is as follows:
MERGE-TYPE=< routine name listed in the first column in tab_merge>

The following is an extract from the tab_expand table:

```
!   1                   2                          3
!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!>

WEB-FULL   expand_doc_hol_bib          MERGE-TYPE=WEB-FULL
```

tab_merge of the tab directory of the holdings library (XXX60):

Column 1 of tab_merge should match column 3 of tab_expand

Column 2 of tab_merge identifies the merge program to be used.

Column 3 contains the merge section to be used from the tab_merge_overlay table.

The following is an extract from the tab_merge table:

```
!   1                 2                              3
!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!>
WEB-FULL   merge_doc_overlay              99
```

tab_merge_overlay of the tab directory of the holdings library (XXX60):

This table is used to define how the holdings record is updated (merged) if it already exists.

The table contains the merge set used in tab_merge column 3.

The following is an extract from tab_merge_overlay table:

```
!1 2 3           4
!!-!-!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
99 1 Y #####
99 2 Y 245##
99 2 Y 1####
```

According to the above example, the holdings record will display all the fields cataloged in the holdings records with an expand of the bibliographic record's title and main entry, as follows:

```
FMT   L HO
LDR   L ^^^^^nx^^a22^^^^^1i^4500
008   L 0207042u^^^^8^^^4001uueng0000000
LKR   L $$aHOL$$lUSM01$$b000000120
CAT   L $$aYOHANAN$$b10$$c20020704$$lUSM60$$h1317
8520  L $$bULINC$$cGEN$$hB819$$i.G3 1968
10010 L $$aGabriel, Leo,$$d1902-
2451  L $$aExistenzphilosophie :$$bKierkegaard, Jaspers,
Heidegger, Sartre. Dial
og der Positionen /$$cLeo Gabriel. --
```

The expand parameters in Column 3 of `tab_expand` can contain up to 10 comma-separated fields (five characters each, including hashes, for example, 85###). If Column 3 is blank, then all fields from the HOL record are added to the ADM record. To exclude a specific field, precede the list of fields with a dash - (for example, -85### -86###).

You can expand the record that is merged into the expanded record. For example: in `expand_doc_lib1_lib2`, it is possible to expand the lib2 record and merge it into the lib1 record. You do this by using a special section in the format "LIB1-LIB2" in the `tab_expand` table of LIB2. For example:

In the case of `expand_doc_hol_bib`, if the section "HOL-BIB" is defined in the `tab_expand` table of the BIB library, the BIB record is first expanded by using this section, and only then are its fields added to the expanded HOL record or merged into it (according to the content of Column 3 of `tab_expand` in the HOL library).

If "MERGE-TYPE=" is used in Column 3 of `tab_expand` for `expand_doc_bib_hol`, it is possible that more than one HOL record will be merged into the BIB record. For example, in `expand_doc_bib_hol`, if more than one HOL record is linked to the BIB record, the first HOL record will be merged into the BIB record; then, the second HOL record will be merged into the already merged BIB record, and so on.

expand_doc_hol_loc_1_a
This program retrieves the item information of all the items attached to the holdings record.

expand_doc_hol_loc_2_a
This program creates an STA field with $aSUPPRESSED-TEMP if all the items attached to the holdings record - retrieved by expand_doc_hol_loc_1_a - are in a temporary location. SUPPRESSED holdings records are not displayed at the head of the items list in the Web OPAC.

Note that both `expand_doc_hol_loc_1_a` and `expand_doc_hol_loc_2_a` are defined in the `tab_expand` table of the holdings library tab directory under the HOL-LOC expand routine.

expand_doc_hol_z30_86x

The `expand_doc_hol_z30_86x` program is used to create holdings fields (V66, V67 and V68) based on the 853-855 (Captions and Pattern) and 853X-855X (Enumeration and Chronology) fields from the holdings record together with the information of the linked item records (from the description field) . One holdings field is created for each item.

V66 is created from 853/853X, V67 is created from 854/854X, and V68 is created from 855/855X.

If the holdings record contains the following fields:

```
85331 L $$av.$$bno.$$u3$$vr$$i(year)$$j(month)$$wt$$ypm01,05,09
853X  L $$a2$$b1$$i2000$$j1$$320000101$$80
```

then the holdings field is created as follows:

```
V6651 L $$av.2:no.1(2000:Jan.)$$80
```

Note that this expand program must be defined in the `tab_expand` table of the holdings library (XXX60).

In addition, you can filter the items by their status. You do this by using the parameters column of the `tab_expand` table. Up to 30 item statuses can be defined (separated by commas). The expand program only creates new fields for the items with the statuses included in the column. If a minus (-) is defined in the first position, then the holdings fields are not created for those items with statuses included in the column.

expand_doc_isbn_13

The expand_doc_isbn_13 expand procedure adds a field with a 13-digit ISBN if the record contains a 10-digit ISBN, and vice versa. Expand_doc_isbn_13 should be added to the "INDEX" section of tab_expand, and the Direct Index (p_manage_05) should be rebuilt in order to apply the expand.

expand_doc_isbn_13_v2

This program is similar to the `expand_doc_isbn_13` program. The difference is that the `expand_doc_isbn_13` program recognizes the following fields as ISBN: 020 for USMARC, 010 for UNIMARC, 021 for DANMARC, and 540 and 634 for MAB and the `expand_doc_isbn_13_v2` program recognizes the following fields as ISBN: 765 - 787 for USMARC and 649 and 770 -787 for MAB (with repeatable subfield $$z).

expand_doc_ismn_13

The expand_doc_ismn_13 routine is used for 13-digit ISMN codes (field 024 indicator 2 in USMARC and 541 in MAB) and their existing 10-digit counterparts. It is used for converting 10-digit ISMN code to 13-digit code and vice versa.

The procedure should be set in tab_expand with the ISMN field index code as a parameter in col.3 (the name of the ISMN field index code from tab00.eng). If no parameter is entered, the default index code is ISMN.

The following is an example of setting expand_doc_ismn_13 in tab_expand:

```
!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!>
INDEX      expand_doc_ismn_13             ISMN
```

expand_doc_issn_isbn

The `expand_doc_issn_isbn` program creates virtual 020/022 fields from the 020/022 fields that contain characters within parentheses. The program removes both the parentheses and the characters.

For example, if the bibliographic record contains the following field:

```
020   L $$a0226123693 (pbk. : v. 2 : alk. paper)
```

Then the program creates the following new virtual field:

```
020   L $$a0226123693
```

expand_doc_join

The `expand_doc_join` program is used with the `tab_expand_join` table of the library's `tab` directory. This table creates a virtual field out of two or more MARC fields. The `tab_expand_join` table determines which fields and which of its subfields should be joined, their order and what the resulting field should be called.

For example, if the library wants to create a headings list of authors and titles, it is possible to define that MARC21 fields 100 and 245 are to be expanded into a new tag (for example, TMP01). You may then send the new virtual tag and 7XX fields that have subfield $t to a common author/title list.

If there are multiple occurrences of the fields, joining is done in pairs. For example, if `tab_expand_join` is set to join 595 and 596 as new field 599, and the record contains "595 a1", "595 a2", "596 b1", "596 b2", "596 b3" the system will create "599 a1 b1" and "599 a2 b2". "596 b3" will be ignored, since it does not have a 595 pair.

Note that you can redirect information to a new subfield in cases where the original field does not have subfields (for example, when joining the MARC 21 001 field).

In the following example, the contents of the 001 field are redirected into subfield $x for the creation of a virtual field (SYS01) created from the 001 field and the 245 field:

```
! 1     2     3     4     5     6     7
!!!!!-!!!!!-!!!!!-!!!!!-!!!!!-!!!!!-!!!!!
SYS01 001         x     245## a     t
```

For example, if the cataloging record contains the following fields:

```
001   L 000003333
24504 L $$aThe Yearbook of Medicine
```

then the new expanded field is created as follows:

```
SYS01 L $$x000000333$$tThe Yearbook of Medicine
```

Entering the TYPE=ALL parameter in Column 3 of the `$data_tab/tab_expand` table creates a virtual field out of two or more MARC fields.

Here is an example of the `tab/tab_expand` table with the definition for this functionality:

```
!   1                   2                              3
!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!>
U39-DOC    expand_doc_join               TYPE=ALL
```

This functionality also uses the used `tab_expand_join` table of the library's `tab` directory. The `tab_expand_join` table determines which fields and which of its subfields should be joined, their order and what the resulting field should be called. The difference is that this program builds a new virtual field for every combination of the field specified in the table. For example, if three fields are to be joined and the first occurs three times, the second occurs three times and the third occurs two times in the record, then 18 new virtual fields are built by this program.

Column 4 in **expand_doc_join** (the new subfield code column) works according to the following methods:

Take one or more subfields from the text and substitute them with the corresponding number of new subfields. For example:

Take *abc* and change to *xyz*

Take all of the text (or several subfields) and change them to one new subfield. For example:

Take *abcd* and change to *x*

Take all but a few subfields (- sign) and change them to one subfield.

Take -*cd* and change to *x*

Take a few subfields (or all the text) and change them to fewer new subfields. It changes all of them to the first one. For example:

Take *abcd* and change to *xy*. It takes the *abcd* and changes it to *x*.

Take all but a few defined subfields and change them to a few new subfields, changing the text to the first new subfield. For example:

Take -*abcd* and change to *xy*. It takes all text except for *abcd* and changes it to *x*.

Here is an example of a possible configuration in `tab_expand_join` which adds virtual fields to a bibliographic record, through the `expand_doc_join` procedure.

```
MAY00 260## abc   xyz   245## abcd  t
MOD00 260## abc   s     245## abcd  t
DEZ00 260## -a    d     245## abcd  t
GOS00 260## abc   gd    245## abcd  t
ROD00 260## -b    os    245## abcd  t
```

expand_doc_join_filter

The expand_doc_join_filter program is used with the tab_expand_join_filter table of the library's tab directory.

This table creates a virtual field out of up to four fields. The tab_expand_join_filter table determines which fields and which of its subfields should be joined, their order, and what the resulting field should be called. It includes the ability to set filter values and the grouping value for joining. Only fields that contain the filter value and have same grouping value are joined.

For more details, see the table: tab_expand_join_filter.

expand_doc_join_permute

This expand procedure uses `tab_expand_join` and creates all the possible permutations of a field.

For example, if `tab_expand_join` contains the following setup:

```
! 1     2     3     4     5     6     7
!!!!!-!!!!!-!!!!!-!!!!!-!!!!!-!!!!!-!!!!! ->
AU700 700## abc   abc   245## ab    fg
AU700 700## abc   abc   740## a     f
```

and if a record has the following lines:

```
000000508 LDR   L 00000cas^^2200589^^^4500
000000508 001   L 000000508
000000508 005   L 20040725104218.0
000000508 008   L 750907c19349999pauqr1p^^^^^^^0uu1a0engdd
000000508 1102  L $$aMusic Library Association.
000000508 24500 L $$aNotes.
000000508 26000 L $$a[Philadelphia, PA, etc.] :$$bMusic Library
Association.
000000508 300   L $$av. :$$bill., ports., facsims. ;$$c24-29
cm.
7001  L $$aO'Meara, Eva Judd,$$eed.
7001  L $$aFox, Charles Warren,$$d1904-$$eed.
7001  L $$aHill, Richard S.$$q(Richard Synyer),$$d1901-
1961,$$eed.
7400  L $$5wid$$aNotes (Online)
7400  L $$aNotes (Journal)
```

then the fields will be expanded in the following way:

```
AU700 L $$aO'Meara, Eva Judd,$$fNotes.
AU700 L $$aFox, Charles Warren,$$fNotes.
AU700 L $$aHill, Richard S.$$fNotes.
AU700 L $$aO'Meara, Eva Judd,$$fNotes (Online)
AU700 L $$aO'Meara, Eva Judd,$$fNotes (Journal)
AU700 L $$aFox, Charles Warren,$$fNotes (Online)
AU700 L $$aFox, Charles Warren,$$fNotes (Journal)
AU700 L $$aHill, Richard S.$$fNotes (Online)
AU700 L $$aHill, Richard S.$$fNotes (Journal)
```

The text taken for the virtual field will be changed according to the filing indicator defined in `tab01.eng`.

expand_doc_join_simple

The `expand_doc_join_simple` program is used with the `tab_expand_join_simple` table of the library's `tab` directory. This table creates a virtual field taking specific occurrences or all occurrences of a field and joins it with specific or all occurrences of another field. The `tab_expand_join_simple` table determines which fields and which of its subfields should be joined, their order and what the resulting field should be called.

Note that the `expand_doc_join` program is useful for indexing, while `expand_doc_join_simple` is useful for creating virtual fields for display.

expand_doc_last_cat

The expand_doc_last_cat program adds a field, named LAS, that is identical to the latest CAT field. Using this expand enables, for example, running retrieve jobs that will fetch only records that have been last updated by a specific cataloger or updated at a specific hour.

expand_doc_link_to_doc

The expand_doc_link_to_doc program adds a field named DRL to a record. This field contains in subfield $a - a direct link to the full view of the record in the Web OPAC.

expand_doc_link_to_ros

For integration between Aleph and Rosetta. In the Aleph Web OPAC, this routine enables the link to the content aggregator in Rosetta for records that are exported to Rosetta. For more information, refer to the Aleph document, *How to Integrate Aleph with Rosetta*.

expand_doc_open_cat

The expand_doc_open_cat program creates a CDATE field from the first CAT field.

expand_doc_own

The expand_doc_own program appends the contents of the OWN field to all fields in the record. The contents are added under subfield $1.

expand_doc_primo_plk

expand_doc_primo_plk provides the linking details from target document to the source. It creates a new field, PLK, which describes a 'DN' link in a document. Using this new field, Primo published documents can be linked the target document to its source (so that the 'UP' and 'DN' linkages are functioning). The fields LKR-TEXT-M and LKR-TEXT-N are created for the PLK field. This field has a limit of 300 characters for each subfield.

expand_doc_ros_id

For integration between Aleph and Rosetta. This expand generates the cms_id tag (Aleph BIB system number). For more information, refer to the Aleph document, *How to Integrate Aleph with Rosetta*.

expand_doc_rotate

The `expand_doc_rotate` program builds a virtual 600 field from subfields $a (personal name) and $t (title of a work) of the MARC21 600 field (subject added entry - personal name). In the new 600 field, subfield $t is sorted before subfield $a.

For example, from 600 $a $c $d $t `expand_doc_rotate` adds 600 $t $a.

expand_doc_section

The `expand_doc_section` program uses the parameters column of the `tab_expand` table to define the section in `tab_expand` (the expand routine - column 1) that should be performed by the program. This program is especially useful when a set of expand programs has to be performed for many instances of the system (for many expand

routines). Instead of defining the group of expand programs for each of the routines, the group can be defined once and then all the different instances can point to it by using the `expand_doc_section` program. Following is an example of the `expand_doc_section` program in `tab_expand`:

```
!   1                   2                              3
!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!->
U39-DOC    expand_doc_section             SECTION1
WEB-FULL   expand_doc_section             SECTION1
WEB-BRIEF  expand_doc_section             SECTION1
GUI-BRIEF  expand_doc_section             SECTION1
GUI-DOC-D  expand_doc_section             SECTION1

SECTION1   expand_doc_bib_loc_usm
SECTION1   expand_doc_yr
SECTION1   expand_doc_type                tab_type_config
```

expand_doc_sort

The `expand_doc_sort` program is used with the `tab_expand_sort` table of the library's `tab` directory. The program sorts the subfields of a specific field according to the sorting order setup in the table.

expand_doc_sort_field

This program sorts a specific field according to the parameters defined in column 3 of the library's `tab_expand` table. In the following example, 260 MARC21 field is sorted according to the contents of subfield $b (the name of the publisher, distributor, and so on.):

`tab_expand` is defined as follows:

```
!   1                   2                              3
!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
U39-DOC    expand_doc_sort_field          260##,b
```

260 fields were cataloged as follows:

```
260   L $$aBogota :$$bOveja Negra,$$c1999.

260   L $$aLondon :$$bHeadline Book Publishing,$$c1994.
```

According to the above definitions of `tab_expand` for `expand_doc_sort_field`, the fields are now sorted as follows:

```
260   L $$aLondon :$$bHeadline Book Publishing,$$c1994.
260   L $$aBogota :$$bOveja Negra,$$c1999.
```

expand_doc_sort_loc_a

This program sorts uniquely the PS1 fields (items + holdings) creating a PST field for each unique PS1. PS1 match for uniqueness is on sublibrary, collection, call number and status.

This program should be used for sites where the items and the holdings records are not linked.

expand_doc_sort_loc_b

This program sorts uniquely the PS1 fields (items) creating a PST field for each unique PS1. PS1 match for uniqueness is on sublibrary, collection, call number,

status, and material type. Note that holdings records that do not have linked items already have their PST field created directly by `expand_doc_bib_loc_1_c`.

This program should be used for sites where the items and the holdings records are linked.

Note that as `expand_doc_sort_loc_a` and `expand_doc_sort_loc_b` are mutually exclusive, it is only possible to use one of these at a time.

expand_doc_split

The `expand_doc_split` program, together with the `tab_expand_split` table of the library's `tab` directory, cuts the content of a tag into separate tags on each occurrence of a subfield, taking all the subfields from one subfield to the next. For example, if subfield $a is set in the table, then $a $b $c $a $a $c is split into:

```
$a $b $c
$a
$a $c
```

Since the split occurs on every occurrence of the subfield, the program creates multiple occurrences of the field. The "split" includes all the data up to the subfield, and all the data from the subfield up to the next occurrence of the subfield, or to the end. The resulting sections of the tag are redirected into virtual "output" tags.

If the `tab_expand_split` table contains the following line:

```
 1   2      3     4
!!!!!-!-----!!!!!-!!!!!
700## t     A700  T700
```

then the following field:

```
700   L $$aMendelssohn-Bartholdy, Felix$$tLider
ohne Worte,$$mpiano
```

is cut into:

```
A700  L $$aMendelssohn-Bartholdy, Felix
T700  L $$tLider ohne Worte,$$mpiano
```

expand_doc_split_external

This program, together with the `tab_expand_external` table of the library's `tab` directory, is used to split the content of a tag, with multiple occurrences of subfields containing external location (856, 505 and so on), into separate tab fields for each occurrence of the designated subfield.

The program duplicates all subfields besides the designated subfield and places them in each new tag field.

Note that the new lines created by the program have the original tag while the original line is suppressed.

The `tab_expand_external` table is used to define fields to be inspected in order to find multiple occurrences of subfields containing external locations, as follows:

```
! 1   2
!!!!!-!-
856## u
505## u
```

For example, if the cataloging record contains the following field:

```
856   L
$$uhttp://jefferson.village.virginia.edu/pmc/contents.all.html$
$uhttp://
lcweb2.loc.gov/ammem/ead/jackson.sgm
```

then the new expanded fields are created as follows:

```
856   L
$$uhttp://jefferson.village.virginia.edu/pmc/contents.all.html
856   L $$uhttp://lcweb2.loc.gov/ammem/ead/jackson.sgm
```

expand_doc_split_sub1

This program is similar to the `expand_doc_split` program. Both programs, together with the `tab_expand_split` table of the library's tab directory, cut the content of a tag into separate tags on each occurrence of a subfield. The difference is that in the `expand_doc_split_sub1` program the resulting paired fields are assigned subfield $1 with a sequence number that shows the original pairing.

For example, if the bibliographic record contains the following fields:

```
7001  L $$aShakespeare, William,$$d1564-1616.$$tPlays.
7001  L $$aDonne, John.$$tSonnets
```

and the tab_expand_split table is defined as follows:

```
! 1   2      3     4
!!!!!-!-----!!!!!-!!!!!
700## t     A700  T700
```

then the expand_doc_split_sub1 program creates the following new fields:

```
A700  L $$aShakespeare, William,$$d1564-1616.$$1001
A700  L $$aDonne, John.$$1002
T700  L $$tPlays.$$1001
T700  L $$tSonnets$$1002
```

This expand routine is required for the correct building of the brief records (Z0101). The building of the brief records relies on subfield $1 in order to pair the split fields. If this program is not used, the brief records for the above example are created as follows:

```
$$aShakespeare, William,$$d1564-1616.$$tPlays.
$$aDonne, John.$$tPlays.
```

When using the `expand_doc_split_sub`1 program, the program that builds the brief records prefers the T700 that has the same subfield $1 as the original A700 that initiated the building of the brief record. The brief records are created as follows:

```
$$aShakespeare, William,$$d1564-1616.$$tPlays.
$$aDonne, John.$$tSonnets.
```

expand_doc_sysno

The `expand_doc_sysno` program is primarily intended for the Z39_SERVER instance in the `tab_expand` table of the library's tab directory. This program creates a virtual SYS field records the Z39.50 server. This program works with the tab04 table of the library's tab directory. The tab04 table is used to set up the specifications for the conversion of the "SYS" tag to a numeric tag . Note that it should always appear under conversion routine 90. The following is an extract from the `tab04` table:

```
90 SYS##            903## N
90 #####            ##### N
```

Note that the last line in this extract must always be present. In addition, note that the program must be defined under the Z39_SERVER entry before the `expand_doc_bib_tab04` program as follows:

```
!   1                   2
!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-
Z39_SERVER expand_doc_sysno
Z39_SERVER expand_doc_bib_tab04
```

expand_doc_type

This program can be used to create a new field according to the specifications defined in a configuration table which is itself a parameter that must be defined in the parameters column of the `tab_expand` table (see Configuration Tables (expand_doc_type). This program, together with the table, can be used, for example, to create a field that contains the format of the record based on the contents of field(s) present in the record (for example, a combination of the LDR and the 008 field). In the following example, the new field contains the string 'FILM' according to a match performed on the values of both the LDR and the 008 field:

```
TYP      Film                LDR   F06-01     EQUAL      g
                             008   F33-01     EQUAL      m
```

In the above example, the TYP field ($aFilm) is created when position 06 of the LDR contains a 'g' and position 33 of the 008 field contains an 'm'. Following is the structure of the new field:

```
TYP   L $$aFilm
```

Note that the name of the configuration table (for example, `tab_type_config`) should be added as a parameter in column 3 of the library's `tab_expand` table.

`expand_doc_type` can be used with repetitive fields. This is especially useful for the EXIST function. For example:

```
!!!!!-!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-!!!!!-!!!!!!!!!!-
!!!!!!!!!!-!!!!!!!!!!!!
TYP   EL Electronic material            856## u          EXIST
```

Note the following:

If the second parameter in `tab_expand` is left blank or set to "Y" as follows:

```
!   1                   2                              3
!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!>
WORD       expand_doc_type                tab_type_config.eng,Y
```

then the program matches the first tag which matches the tag+subfield condition. Accordingly, the EQUAL, N-EQUAL or MATCH functions cannot be used for repetitive fields with the same subfield. For example:

```
TYP      Publisher                   260## b         EQUAL      Oclc
```

The line above does not work with a record such as:

```
000000000 L 26001 $$aNew York$$bRandom House
000000000 L 26001 $$aNew York$$bOCLC
```

since it only takes the first line.

If the second parameter in `tab_expand` is set to "N" as follows:

```
!   1                   2                              3
!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!>
```

```
WORD        expand_doc_type                    tab_type_config.eng,N
```
then the program will attempt to find matches with every occurrence of the repetitive field and subfields in the record.

For example:

`tab_type_config` includes the following lines:

```
! 1   2            3                        4       5         6            7
!!!!!-!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-!!!!!-!!!!!!!!!-!!!!!!!!!-
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-!
LOC      LOC-1                         852## b          EQUAL      WID
LOC      LOC-2                         852## b          EQUAL      HIL
LOC      LOC_3                         852## b          EQUAL      MED
LOC      LOC-4                         852## b          EQUAL      LAW
```

For a record that includes the following fields:

```
852   L $$bLAW
852   L $$bMED
852   L $$bHIL
852   L $$bWID
```

The result of the expand will be:

```
LOC   L $$aLOC-1
LOC   L $$aLOC-2
LOC   L $$aLOC_3
LOC   L $$aLOC-4
```

It is also possible to create TYP fields for multiple subfields occurrences. In the example below the first character is “S” (for subfield), the second character is the subfield itself (b), and the third character is the subfield occurrence (1-9,A). To enable the expand to check the match for multiple subfield occurrences, set the third character to "A" (all). For example:

! 1   2   3   4   5   6   7

!!!!!-!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-!!!!!-!!!!!!!!!-!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-!

TYP   SHB   042## SbA   MATCH   SHB

expand_doc_uni_merge

The `expand_doc_uni_merge` program is used by UNIMARC libraries to merge linked records and display them together. For example, for displaying linked records in brief format, the following line should be present in `tab_expand`:

```
!   1                    2
!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
WEB-BRIEF  expand_doc_uni_merge
```

expand_doc_union_add_852

The expand_doc_union_add_852 program adds, in a union catalog, a dummy 852 field containing the value of SID $$b in cases when a record has only 856 fields without 852. This program should reside in to the PRE-MERGE section in `tab_expand`.

Note that there must be a matching Z124 so that it will be displayed.

expand_doc_union_exclude_lib

The expand_doc_union_exclude_lib routine faciliates preventing Web OPAC download of union records from non-authorized users. Column 3 of tab_expand is used when setting up this expand routine to set which contributing library is to be removed by this expand routine. The SID,852,856 and 866 fields of the libraries given

as parameter are deleted. If more than one library code is to be used in column 3, the library codes must be separated by a comma.

expand_doc_yr
This expand program builds a virtual YR field that contains the publication date (year) based on the parameters entered in column 3 of the `tab_expand` table. Following is the structure of the YR field:

```
YR  $a [year]
```

In the following example, the `tab_expand` table is set up so that the year is taken from the 008 MARC 21 field. If the 008 field has no publication date, the contents expanded into the new YR field are taken from subfield $c of the 260 MARC 21 field:

```
!   1                   2                              3
!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
U39-DOC    expand_doc_yr                  008,260##c
```

If the publication year is 2001, for example, then the new field is built as follows:

```
YR    L $$a2001
```

This program can also take the year from the 100 UNIMARC field:

```
!   1                   2                              3
!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
U39-DOC    expand_doc_yr                  100
```

Note:
In addition to the above list of expand routines, it is also possible to use external expand routines. The external expand routines can be written in any programming language and can be executed without being linked to Aleph. These expand programs are defined in tab_expand and can be used for extended bibliographic information. The external program must reside in $aleph_exe and should not have an extension.

9.3 Expand-Related Tables

9.3.1 Configuration tables (expand_doc_type)

The **tab_type_config** table in the library's `tab` directory is a sample of a table that can be used with the `expand_doc_type` program. The table defines specifications for the creation of a new field. Additional similar tables can be added, after which they can be listed as parameters in the `tab_expand` (in col.3) table.

In the following example, the new virtual field contains 'FILM' according to the values of both the LDR and the 008 field:

```
TYP      Film                LDR   F06-01     EQUAL      g
                             008   F33-01     EQUAL      m
```

In the above example, the TYP field ($aFilm) is created when position 06 of the LDR contains a 'g' and position 33 of the 008 field contains an 'm'.

Following is a sample from a configuration table:

```
! 1   2            3            4        5         6         7
8
!!!!!-!!-!!!!!!!!!!!!!!!!!!!!-!!!!!-!!!!!!!!!!-!!!!!!!!!!-
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-!
982      Media                LDR   F06-01     EQUAL      [g,k,r,o]

982   BK Book                 LDR   F06-01     EQUAL      a

982      Film                 LDR   F06-01     EQUAL      g
                              008   F33-01     EQUAL      m

982      Videorecording       LDR   F06-01     EQUAL      g
                              008   F08-01     EQUAL      v
```

Note that the tables used by the `expand_doc_type` program are language-sensitive. If, for example, the `tab_expand` table is set up as follows:

```
!   1                   2                              3
!!!!!!!!!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!>
WORD       expand_doc_type                tab_type_config
```

then the system looks for the matching `tab_type_config` in the following order based on the language:

$data_tab/tab_type_config.<lng>

$data_tab/tab_type_config.<control_lng> (as defined in aleph _start)

$data_tab/tab_type_config

Key to the configuration tables (for example, tab_type_config):

Column 1 - Target tag
Target field created with the contents of column 2 and/or column 3 by the expand_doc_type.

Column 2 - Format code
Format code, for example, BK (for book). The value entered in this column is expanded into subfield $a of the new field created by `expand_doc_type`. If the column is left blank, then the format name (value of column 3) is added to subfield $a of the new field. For example, if the table contains the following line:

```
      TYP   BK Book                 LDR   F06-01     EQUAL      a
```

then a new TYP field with the following structure is added when position 06 of the LDR field contains an 'a':

```
TYP   L $$aBK$$bBook
```

If this column is left blank, the new field will be created/expanded as follows:

```
TYP   L $$aBook
```

Column 3 - Format name

Format name, for example, Book. If a format code is present (column 2), then the format name is added/expanded into subfield $b of the new field. If no format code is defined, then the format name is added/expanded into subfield $a of the new field. For example, if the table contains the following line:

```
TYP   BK Book                 LDR   F06-01     EQUAL      a
```

then a new TYP field with the following structure is added when position 06 of the LDR field contains an 'a':

```
TYP   L $$aBK$$bBook
```

If the format code column is left blank, the new field is created/expanded as follows:

```
TYP   L $$aBook
```

Column 4 - Field tag

Field from the record used to determine the material type that is expanded into the new field. In the following line, the LDR (position 06 with 'a') is used to define that the record is for a book:

```
TYP   BK Book                 LDR   F06-01     EQUAL      a
```

Column 5 - Subfield(s) or fixed field position

This column contains the subfield codes or the fixed field positions (of the field defined in column 4) to be checked. In the following line, the program checks position 06 of the LDR:

```
TYP   BK Book                 LDR   F06-01     EQUAL      a
```

In the following line, the program checks subfield $a of the 490 field:

```
TYP    Thesis          4901# a     MATCH     [masters*,education*]
```

You can specify positional conditions in this column by the following syntax:

S<subfield><occurrence>FNN-NN

For example

```
!!!!!-!!-!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-!!!!!-!!!!!!!!!!-!!!!!!!!!!-
!!!!!!!!!!!
TYP   AB Book                          100   Sa1F06-01  EQUAL      x
```

This example shows that if in field 100, the first subfield is *a*, which occurs once, and position 6 length 1 contains *x*, then an expanded TYP field will contain *AB book*.

Column 6 - Match criteria
This column is used to define the match criteria in relation to the contents of the subfield or the fixed field positions defined in column 5. Following are the available options: EQUAL, N-EQUAL, EXIST, N-EXIST, MATCH and N-MATCH.

Usage:

EQUAL (N-EQUAL) - checks for direct match

EXIST (N-EXIST) - checks if the field exists without checking the field contents. For example, if a record has a 027 MARC 21 field, then the record is a technical report (contents are irrelevant).

MATCH (N-MATCH) - checks for a match that is not case-sensitive

Column 7 - Contents
This column contains the contents of the field or of the fixed field position used for matching (according to the match criteria defined in column 6). Use [] to enclose multiple values for matching (up to 15 different comparison values). The relationship between the values is of type OR. In the following line, the match is based on values 'e' or 'f' of position 06 of the LDR field:

```
      TYP      Map                    LDR   F06-01     EQUAL      [e,f]
```

Column 8 - Case-sensitive matching
Case-sensitive matching flag. Values are:

Y = Matching is case-sensitive

N = Matching is not case-sensitive

Note that if this column is left blank, the default is matching which is not case-sensitive.

9.3.2 tab_expand_split

The **tab_expand_split** table is used to set up the specifications for the `expand_doc_split` program. This program splits a field into separate parts, splitting on the subfield specified in column 2 of the `tab_expand_split` table. The split occurs on every occurrence of the subfield, thereby creating multiple occurrences of the field.

The "split" includes all the data up to the subfield, and all the data from the subfield up to the next occurrence of the subfield, or to the end. For example, if subfield $a is set in the table, then $a $b $c $a $a $c split into:

```
$a $b $c
```

```
$a
$a $c
```
The following is a sample of the `tab_expand_split` table:

```
! 1   2      3     4
!!!!!-!-----!!!!!-!!!!!
260## b     PLA   PUB
700## t     100   240
```

Key to the tab_expand_split table:

Column 1 - Tag and Indicators
Contains the tag and indicators of the document field to be treated by the expand program. # can be used for the third to fifth positions to indicate truncation of numeric additions to the field code (for example, 245## is used for 2451, 2452, 24501, and so on).

Column 2 - Subfield
Contains the subfield to be split on.

Column 3 - "Up-to" Tag
Output field tag and indicators for text up to the breaking subfield.

Column 4 - "After" Tag
Output field tag and indicators for text after the breaking subfield.

Based on the sample table, the following field:

```
260   $$aBoston: $$bLittle, Brown, and company,$$c1933.
```

is cut into:

```
PLA   $$aBoston:
PUB   $$bLittle, Brown, and company,$$c1933.
```

9.3.3 tab_abbrev

The `tab_abbrev` table used is in conjunction with the `expand_doc_fix_abbreviation` program.

The `expand_doc_fix_abbreviation` program is used to add a duplicate virtual field to the document where the shortened form (abbreviation) of a word or a phrase is changed to the whole word or phrase. In actual fact, the routine can be used to replace any text string in a bibliographic record with a different text string. The `tab_abbrev` table contains the list of "abbreviations" and the whole forms into which the shortened form is to be changed in the new virtual field added to the record by the `expand_doc_fix_abbreviation` program. The text exchange is defined per field tag (including indicators).

Note that this program can be also used as a fix program (using `tab_fix` in place of

tab_expand). The difference is that instead of adding virtual fields (for indexing or/and display), when used as a fix program, the fields with the whole forms are actually added to the record.

Following is a sample of the tab_abbrev table:

```
! 1   2         3                              4
!!!!!-!-!!!!!!!!!!!!!!!!!!!-
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
225## Y 1st                  FIRST
043## Y u-at---              Australia
043## Y u-at-ne              Australia New South Wales
```

Key to the tab_abbrev table:

Column 1 - Field Tag and Indicators
Contains the tag and indicators of the document field to be treated by the expand program. Field tags can be grouped by using # as a wildcard (for example, 2#### can be used to include all field tags that begin with "2").

Column 2 - Usage Code
Select the appropriate usage code. The following values are available:

X: Defines that the tag is not to be considered at all for any abbreviation fixing. Therefore, an X line should not include text in columns 3 and 4.

N: Defines that the text exchange should be ignored for the particular tag or tag grouping. An N line is necessary only if it is followed by a Y line which uses #. For example, the following line in the table changes Ft. to FORT in all fields that begin with 2:

```
2#### Y Ft.                   FORT
```

If the table contains the following:

```
245## N Ft.                   FORT
2#### Y Ft.                   FORT
```

the program changes Ft. to FORT in all fields that begin with 2, except for field tag 245.

Y: Defines that the text exchange should occur.

Column 3 - Abbreviation
Contains the text that should be changed in the new field added to the record.

Column 4 - Expanded Form
Contains the expanded form for the abbreviation in column 3.

Based on the sample table, the following document:

```
043   L $$au-at-ne
1001  L $$aWilliams, David
2451  L $$aThe 1st man
```

is expanded into:

```
043   L $$au-at-ne
1001  L $$aWilliams, David
2451  L $$aThe 1st man
043   L $$aAustralia New South Wales
2451  L $$aThe FIRST man
```

9.3.4 tab_expand_duplicate_field

The `tab_expand_duplicate_field` table is used to set up the specifications for the expand_doc_duplicate_field program. This program is used to duplicate a field, assigning a new field tag plus indicators.

The following is a sample of the `tab_expand_duplicate_field`

```
! 1   2
!!!!!-!!!!!
260&335;# IMP
```

The expand_doc_duplicate_field procedure and expand_doc_duplicate table can be used in order to overcome the problem created when using expand_doc_split, which does not retain the source field. When expand_doc_split is based on a field created by the expand_doc_duplicate_field program, the source field is retained. In the above example, the IMP field can be later processed by expand_doc_split without losing the original 260 field.

Key to the tab_expand_split table:

Column 1 - Input Field Tag and Indicators
Contains the tag and indicators of the document field to be duplicated by the expand program.

Column 2 - Output Field Tag and Indicators
Contains tag and indicators after duplication.

Based on the sample table, the following field:

```
2600 L $$aBoston, :$$bRoberts Bros
```

Is duplicated into:

```
IMP   L $$aBoston, :$$bRoberts Bros
```

9.3.5 tab_expand_external

The `tab_expand_external` table is used together with the `expand_doc_split_external` program.

The `expand_doc_split_external` program is used to split the contents of a tag with multiple occurrences of subfields containing an external location (856, 505 and so on) into separate tab fields for each occurrence of the designated subfield.

The `tab_expand_external` table is used to define fields to be inspected in order to find multiple occurrences of subfields containing external locations, as follows:

```
! 1   2
!!!!!-!-
856## u
505## u
```

Key to the tab_expand_external table:

Column 1 - Input Field Tag and Indicators
Contains the tag and indicators of the document field to be inspected by the expand program.

Column 2 - Output Field Tag and Indicators
Contains subfield tag occurring multiple times in the field.

Based on the sample table, in the case of the following field:

```
5054  L $$:a505 FORMATTED CONTENTS
NOTE$$:uhttp://lcweb.loc.gov/marc/bibliograph
ic/ecbdnot2.html#mrcb555$$:uhttp://www.loc.gov/standards/mets
```

the new expanded fields are created as follows:

```
5054  L $$:a505 FORMATTED CONTENTS
NOTE$$:uhttp://lcweb.loc.gov/marc/bibliographic/ecbdnot2.html#mrcb555

5054 L $$:a505 FORMATTED CONTENTS
NOTE$$:uhttp://www.loc.gov/standards/mets
```

9.3.6 expand_doc_bib_z30

The **expand_doc_bib_z30** table is used to define the information from the item record that is expanded by the expand_doc_bib_z30 program. The table defines the following: which fields are taken from the item record; in which cases these fields should be taken (for issues, for copy items, and so on,); in which subfields of the

expanded field the information should be stored; and for some specific item fields, whether the codes should be replaced by names (for example, the sublibrary code can be replaced by the sublibrary name).

9.3.7 expand_doc_bib_z403

The expand_doc_bib_z403 table is used to define the information from the object's data information that is expanded by the expand_doc_bib_z403 program. The table defines the following: which fields are taken from the object's data record (Z403); in which subfields of the expanded field the information should be stored; and for some specific fields of the object's data record, whether the codes should be replaced by names (for example, the sublibrary code can be replaced by the sublibrary name).

Note that column two of this table usually contains the field from the Z403 record (for example, z403-sub-library), in order to create a valid 856 URL field, the column should contain url and the expanded field should be set in tab_expand to be 856. This option should only be used in special cases such as the X and Z39.50 servers.

9.4 Indexing Expand Fields (Virtual Fields)

In order to index, follow these steps:

Set up the appropriate expand table. For example:

tab_expand_extract

```
! 1   2   3
!!!!!-!-!!!!!
650## a SUBJ
```

Add the expand field to tab11.

Call the expand program in the ACC/ WORD/ INDEX entry of tab_expand:

For keyword indexes, the expand program must be added to the WORD entry in tab expand.

For headings, the expand program must be added to the ACC entry.

For direct indexes, the expand program must be added to the INDEX entry, and for sort indexes, it must be added to the entry SORT-DOC.

For display, it must be added to WEB-FULL, WEB-FULL-1, WEB-BRIEF, GUI-DOC, GUI-BRIEF.

Example:

tab_expand

```
WORD        expand_doc_fmt_mgu
WORD        expand_doc_join
!
ACC         expand_doc_bib_loc_usm
ACC         expand_doc_bib_ndu
ACC         expand_doc_join
```

```
!
INDEX       expand_doc_extract
INDEX       expand_doc_bib_loc_usm
SORT-DOC    expand_doc_bib_loc_usm
!
WEB-BRIEF   expand_doc_join_simple
!
```

9.4.1 tab_expand_extract

This table defines the extraction of subfields for indexing. For example, if you want to create a headings list of chronological subdivisions for subject added entries - topical terms, you can define that MARC21 650 field, subfield $y is to be expanded into a new tag (for example, y650). The virtual field may then be indexed or displayed.

Note that the fourth column in the tab_expand_extract table can be used to specify the number of subfield occurrences for which the new virtual field is created. For example, you can define that only the first occurrence of subfield $y in the 650 field should be used for the creation of the new field. The following is an example of extraction from the table:

```
! 1   2   3   4
!!!!!-!-!!!!!-!
650#:#y CHRON  1
```

Key to the tab_expand_extract table

Column 1 - Input field tag and Indicators
Tag and indicators of the document field containing the subfield to be extracted by the program.

Column 2
Contains the subfield to be extracted by the program and expanded into a new field tag.

Column 3 - Output Field Tag and Indicators
Contains the new field tag for indexing

Based on the sample table, the following subfield y:

```
650-0 L $$y19th century$$aProtestants$$zFrance.
```

Is extracted into the virtual field tag CHRON as follows:

```
CHRON L $$y19th century
```

9.4.2 tab_expand_join

This table creates a virtual field out of two or more MARC fields. The `tab_expand_join` table determines which fields and which of its subfields should be joined, their order and what the resulting field should be called.

For example, if you want to create a headings list of authors and titles, you can define that MARC21 fields 100 and 245 are to be expanded into a new tag (for example, AUTIT). You can then send the new virtual tag to an author/title list. The virtual field is used for indexing purposes.

Following is a sample from the `tab_expand_join` table:

```
! 1     2     3     4     5     6
!!!!!-!!!!!-!!!!!-!!!!!-!!!!!-!!!!!
AUTIT 100## a           245## abcd
```

Key to the tab_expand_join table

Column 1 - New virtual field tag and indicators

Column 2 - First tag for building the new virtual tag

Column 3 - Subfields to take for the match
If this column is empty, all subfields are taken. If you want a selection of subfields, enter the subfield codes.

Column 4 - New subfield code

Column 5 - Second tag for building the new virtual tag

Column 6 - Subfields to take for match
If this column is empty, all subfields are taken. If you want a selection of subfields, enter the subfield codes.

Note

If there are multiple occurrences of the fields, joining is done in pairs. For example, if tab_expand_join is set to join 595 and 596 as the i new field 599, and if the record contains "595 a1", "595 a2", "596 b1", "596 b2", "596 b3", the system will create "599 a1 b1" and "599 a2 b2". "596 b3" will be ignored, since it does not have a 595 pair.

Based on the sample table, the following fields:

```
10010 L $$aOrcibal, Jean.24510 L $$aLouis XIV et les protestants,
:$$b"La cabale des accommodeurs de reli
```

Are concatenated into the virtual field tag AUTIT as follows:

```
AUTIT L $$aOrcibal, Jean.$$aLouis XIV et les protestants,
$$b"La cabale des accommodeurs de religion", la caisse des
conversions, la revocation de l'dit de Nantes.
```

9.4.3 tab_expand_join_simple

This table creates a virtual field taking specific occurrences or all occurrences of a field and concatenates it with specific or all occurrences of another field. The `tab_expand_join_simple` table determines which fields and which of its subfields should be concatenated, their order and what the resulting field should be called. The virtual field is used for display purposes.

Following is a sample from the `tab_expand_join_simple` table:

```
! 1     2   3    4     5     6    7   8     9
!!!!!-!!!!!-!!-!!!!!-!!!!!-!!!!!-!!-!!!!!-!!!!!
ATS02 100## AA a     a     245## AA       y
```

Key to the tab_expand_join_simple table

Column 1 - New virtual field tag and indicators

Column 2 - First tag for building the new virtual tag

Column 3 - First tag occurrence; nn for index, AA for All

Column 4 - Subfield to take for match

Column 5 - New subfield code

Column 6 - Second tag for building the new virtual tag

Column 7 - Second tag occurrence; nn for index, AA for All

Column 8 - Subfield for match

Column 9 - New subfield code

Based on the sample table, the following fields:

```
10010 L $$aOrcibal, Jean.
24510 L $$aLouis XIV et les protestants, :$$b"La cabale des
accommodeurs de religion", la caisse des conversions, la revocation
de l'dit de Nantes.
```

Are concatenated into the virtual field tag ATS02 as follows:

```
ATS02 L $$aOrcibal, Jean.$$aLouis XIV et les protestants,
:$$b"La cabale des accommodeurs de religion", la caisse des
conversions, la revocation de l'dit de Nantes.
```

10 Other Indexes

This chapter includes the following sections:

Update Short Bibliographic Records (manage-07)

Update Sort Index (manage-27)

Update Indexes for Selected Records (manage-40)

10.1 Update Short Bibliographic Records (manage-07)

A short bibliographic record is an abbreviated version of the bibliographic record in standard Oracle table format. It contains up to six fixed (system-defined) fields (year, call number/call number key, author, title, imprint and ISSN/ISBN) limited to 100 characters each (except for the call number key field that can be up to 80 characters long). Additionally, it can contain up to fifteen user-defined fields, each limited to 500 characters.

The purpose of a short bibliographic record is to provide bibliographic information in an effective and timely manner, particularly for instances where bibliographic information is an adjunct to administrative information.

A short bibliographic record is built by the system according to the definitions in the `tab22` table.

When to Update Short Bibliographic Records

Run this service after making any change in the tab22 table that affects the short bibliographic records.

Short bibliographic records must be updated if a large number of records have been uploaded into the database in the "partial" mode, since the system does not build short bibliographic records automatically for these documents.

10.2 Update Sort Index (manage-27)

This service updates the Sort Index of the database.

The Z101 table contains sorting keys. When a list of brief records is displayed in the Web OPAC, the Z101 records are used to arrange the list in a specific order. The fields that are used for building sorting keys are defined in the library's `tab/tab_sort` table in the library's tab directory.

10.3 Update Indexes for Selected Records (manage-40)

This service writes the requested document numbers in the Z07 Oracle table, after which the records are re-indexed through the library's usual updating process (ue_01).

When to Run This Service

Run this service when you need to re-index selected document records.

11 Preparation for Index Jobs

p_manage_01, p_manage_02, p_manage_07 (for z00r), and p_manage_32 can require a large amount of disk space when they are run on large databases if this is the first time they are being run in the specific library and may take a long time to run.

11.1 Clean temp/scratch Directories

Make sure that the `xxx01 $data_scratch`, the `xxx01 $data_files`, and the `$TMPDIR` directories are cleaned of any extraneous, temporary files. If, based on calculations in the preceding disk space section, the disks on which the `$data_scratch`, `$data_files`, or `$TMPDIR` reside may not have enough space, consider moving them temporarily to a different location.

11.2 Check Oracle Space

You can use UTIL A/17/11 (Check Space Utilization of Oracle Tables) to make sure that the relevant files (Z97/Z98 for Words, Z01/Z02 for Headings, and Z11 for Direct index) are nowhere near their maximum number of extents (505, as delivered by Ex Libris) or their maximum tablespace sizes. If they are, the extent sizes specified in the `xxx01/file_list` or the tablespace(s) sizes should be increased prior to running the job.

11.3 Cancel Jobs Which Might Interfere

manage_nn index jobs may need to run when backup jobs or other jobs normally run. You need to make sure that jobs which would interfere with the index job are cancelled. The mechanisms for automatic running are the ALEPH job daemon and the Unix cron. The ALEPH job daemon can be cancelled via UTIL E/15 or individual jobs can be commented out via UTIL E/16. Consult your Unix systems administrator in regard to cron jobs which might interfere.

12 Parallel Indexing

Parallel indexing is used to rebuild an OPAC index parallel to the online ALEPH system without downtime while the index is being created. This process also lets you change indexing parameters and check the results without losing current indexes.

The indexing is carried out in a separate library. This library is set up with a pointer to the document records in the actual library, and indexes are located in the indexing library. After indexing has been completed, a pointer is created in the actual library to the index in the indexing library.

In the examples in the following sections, USM01 is used as the actual library, and USM21 is used as the indexing library.

To start the indexing process in the USM21 library, assume that all index tables (for example, for Word tables this would be Z97, Z98, and so on) are defined as local in the files list, and the documents file as a logical synonym to USM01. You can now run the indexing job (for example, p_manage_01) in USM21. It reads records from USM01 via the logical synonym, but creates index tables locally. You can also change the index setup in the indexing library to create different index codes or to use different filing procedures.

After the index is built successfully, check it in USM21 using Web OPAC.

Finally, if you want to switch to the new index, create a logical synonym from USM01 to USM21 for all relevant tables. Thus there is no downtime whatsoever.

The re-indexing of word (W-nnn) and direct (IND) indexes is an autonomous process that does not require any other indexing. Re-indexing the headings (ACC) index, on the other hand, requires a series of indexing jobs.

The following are the steps that should be taken to activate parallel indexing:

Step 1: Open a Library
Open a new BIB library, parallel to the library that is going to be indexed.

Step 2: Add the Indexing Library to library_relation
The library_relation table in the /alephe/tab directory defines relationships among various libraries. For parallel indexing, a PID relationship must be defined, as in the following example:

PID USM21 USM01

In addition, the relationship between the indexing library and the ADM and HOL libraries must be defined in exactly the same way as the relationship between the actual library and its related ADM and HOL libraries. For example:

ADM USM01 USM50 USM51
HOL USM01 USM60

ADM USM21 USM50 USM51
HOL USM21 USM60

PID USM21 USM01

Step 3: Adjust the Library's file_list
In the root directory of each ALEPH library, there is a configuration table called file_list. This table lists all of the library's Oracle tables, their size, extents and location. In this configuration table, you can define that the library uses an Oracle

table from a different library, instead of its own Oracle table. You do this by setting a pointer to the other library, using a Logical Symbol (LS) definition.

Initially, in the root directory of the new library, there should be a copy of the actual library's file_list. At this stage, the file_list should contain all the Oracle tables listed in the actual library's file_list, using the same definitions. Later on, the definitions are changed, in both the actual and the indexing library, as required.

In the indexing library, the sequence numbers table, Z52, must always be local:

```
TAB z52          10K          10K          ts0
IND z52_id       10K          10K          ts1
```

In the indexing library, you must always use a logical synonym for the bibliographic documents table, Z00 as well as Z103 table:

```
LS  z00          usm01
LS  z103         usm01
```

In addition, the TAB and IND lines for the z00 and z103 need to be commented out (by placing "!" in front of them).
Before initiating parallel indexing, the above tables should be dropped in the indexing library (the parallel library).
Before doing the SQL drops for this (shown further below), do the following select:

```
 >>s+ usm21
SQL-USM21> select count(*) from  Z00 ;
SQL-USM21> select count(*) from  Z103 ;
```

The result to both should be 0. If it is not, it indicates that you are either:
(1) not in the parallel library or
(2) LS-ed from the parallel library to the production.
Quit immediately.
In the case of #1, do "s+" to the correct, parallel library.
In the case of #2, do util a/17/5/1 to confirm that the synonyms already exist. If so, you do not need to do anything more. The fact that the synonyms exist indicates that the tables have been dropped. (Oracle does not let you create the synonym when the table exists in the parallel library.)
If the result of the "select count" is 0, then continue with the SQL commands for doing the drop:

```
SQL-USM21>drop table Z00;
SQL-USM21>drop table Z103;
```

After changing the file_list and dropping the tables, create the logical synonyms (UTIL A/17/5/2) in the parallel library.
Note: If you are building Words Index and you use synonyms (Z970), all the said above for z00 and z103 is true also for z970.
Do not forget to check if there is enough free space to build the index in the parallel library tablespace (util o/14/1). You can see the size of the index tables in the actual (USM01) library with util a/17/11/2. (Note that the latter result is in KB whereas the first is in MB). The new tables require an equal amount of space in the parallel library tablespaces (-- unless you change the number of indexes or fields indexed).

Step 4: Adjust the Library's Z52 Table

Before initiating parallel indexing, make sure that the following counters are defined as listed. Use UTIL/G/2 to add missing counters, and update the counter values.

The required values are:
last-doc-number (set to the same value as last-doc-number of the actual library)
last acc-number (set to 0)
last-long-acc-number (set to 0)
last-similar-acc-num (set to 0)
last-word-number (set to 0)

Step 5: Set up Indexing Configuration Tables

Define the setup of the indexing tables in the tab directory of the indexing library. You can choose to copy the tables that are used for indexing in the actual library into the tab directory of the indexing library, or you can use the path_convert configuration table to direct the system to the actual library's configuration tables. This option is feasible only if you do not want to change the indexing setup.
You may change $data_tab values in the indexing library in order to improve the indexing. If you do this, be sure to copy these changed tables to the actual library in Step 9.

Step 6: Save Interim Indexing Updates

While the indexing jobs are running, new and updated records in the actual library are indexed, through the ue_01 and Z07 mechanism, on the old indexes. These Z07 records must be saved for re-indexing after the new indexes have been built. In order to save these records, before running the indexing process, activate the E/5/1 utility in the actual library. UTIL E/5 lets you create a history table for Z07 entries handled by ue_01. These stored entries may be later used to re-execute ue_01 on the same records.

Step 7: Run the Indexing Jobs

Word Indexes

Run p_manage_01 to rebuild the Word Index of the database.
Direct Indexes
Run p_manage_05 to rebuild the Direct Index of the database.
Headings Indexes
Optional: Run p_manage_102; to pre-enrich the bibliographic headings index, based on the Authority Database.
Run p_manage_02.
Start UE-08 to build the bibliographic heading - Authority Record connection.

Note: If you have run p_manage_102, starting UE-08 at this stage is not necessary.
Run p_manage_105 in the AUT libraries to add untraced references.
Run p_manage_17 to alphabetize long headings.
Run p_manage_35 to create brief records (z0101) .
Run p_manage_32 to build counters for logical bases.
Note: Since v17, p_manage_32 can be run in the parallel library.

Note: When rebuilding headings (browse) indexes, you must run the additional indexing processes listed above.
If your AUT database does not include untraced headings, there is no need to run p_manage_105.

If you do not have logical bases, or if you have not set "Y" in column 8 of tab_base.lng for any of the bases, then there is no need to run p_manage_32.

If you are not using brief records, you do not have to run p_manage_35.
Other Indexing Jobs to Run
Run p_ manage-07 to update short bibliographic records.
Run p_manage_27 to update the sort index.

Step 8: Check the New Index
Add the indexing library to /alephe/tab/tab_base.lng and add the library to the base list for the Web OPAC (/alephe/www_f_lg/base_list). Access the WEB OPAC, choose the indexing library, and check the new index. If all appears satisfactory, continue with Step 9.
Note: The location does not display in the OPAC Brief or Full displays when performing this test. (This is because there is no ADM or HOL for the parallel library.)
If this run of the parallel indexing is just for the Keywords, then the Browse and the Browse links in the Full display do not work in performing this test. If this run of the parallel indexing is just for the Browse, then the Keyword does not work in performing this test.

If your site is using Union Catalog or Union View, note that they do not work in performing this test unless you add LS's for the z120 and z127.

Step 9: Applying the New Indexes
When the indexing has been completed, stop all running daemons in the actual library (UTIL E).

Option 1: Logical Synonyms
This option uses logical synonyms to point from the actual library to the indexing library Oracle tables in order to apply the new indexes.

The next step required is switching from the current (old) index to the new index. Create a pointer from the actual library Oracle table to the Oracle table in the indexing library, by changing the definition in the actual library's file_list to a logical synonym.

The following example shows the new setup after re-indexing headings:

Replace:

TAB z01	2M	1M	ts0
IND z01_id	1M	1M	ts1
IND z01_id2	300K	100K	ts1
IND z01_id3	200K	100K	ts1
IND z01_id4	200K	100K	ts1

```
IND z01_id5        200K        100K        ts1
IND z01_id6        200K        100K        ts1
```

With:

```
LS  z01          USM21
```

Replace:

```
TAB z02          400K        100K        ts0
IND z02_id       300K        100K        ts1
IND z02_id1      400K        100K        ts1
```

With:

```
LS  z02          USM21
```

The following example shows the new setup after re-indexing words:

Replace:

```
TAB z95          1M          1M          ts0
IND z95_id       1M          1M          ts1
TAB z97          2M          1M          ts0
IND z97_id       1M          1M          ts1
IND z97_id1      1M          1M          ts1
IND z97_id2      1M          1M          ts1
IND z97_id3      1M          1M          ts1
TAB z98          3M          1M          ts0
IND z98_id       2M          1M          ts1
TAB z980         1M          1M          ts0
IND z980_id      1M          1M          ts1
```

With:

```
LS  z95          USM21
LS  z97          USM21
LS  z98          USM21
LS  z980         USM21
```

Before doing the SQL drops for this (shown further below), do the following select:

```
>>s+ usm21
SQL-USM21> select count(*) from   Z01   ;
SQL-USM21> select count(*) from  Z02 ;
```

Check that these tables are not empty.

Drop the relevant Z tables (mentioned above) in the actual library by using the SQL command, as in the following example:

```
>>s+ usm01
```

SQL-USM01>drop table Z01;
SQL-USM01>drop table Z02;
etc.
Then create logical synonyms to the indexing library, using UTIL A/17/5 in the actual library.

Note: If you use this option, the next time you want to re-index, you must create an additional indexing library (in order that both the BIB documents library and the indexing library remain unlocked).

Option2: Oracle Import
Installations that have Oracle DBA expertise can choose to copy the new indexes (that is, the Oracle tables) from the indexing library to the actual library.

Whether Option 1 or Option 2 is chosen, the following actions must be done:
When the new index is a result of the p_manage_02 process, update the last-acc-number and the last-similar-acc-number counters in the actual library (using UTIL G/2) to the same value as the counter in the indexing library.
If the p_manage_02 service has been run in "Update headings index" procedure and Duplicate Mode: Yes; update the last-long-acc-number counter in the actual library (using UTIL G/2) to the same value as the counter in the indexing library.
When the new index is a result of the p_manage_01 process, update the last-word-number counter in the actual library (using UTIL G/2) with the same value as the counter in the indexing library.
When the new index is a result of the p_manage_35 process, update the last-z0101-sequence counter in the actual library (using UTIL G/2) with the same value as the counter in the indexing library.
When the new index is a result of the p_manage_17 process, update the last-long-acc-number counter in the actual library (using UTIL G/2) with the same value as the counter in the indexing library.
If you have changed $data_tab tables in the indexing library in order to improve the indexing (see Step 5, above), copy these changed tables to the actual library. In case some of these files have an earlier timestamp than the ones they are replacing, do util x/7 to clean out the utf_files.
The last action in this section of the re-indexing process is restarting the daemons (UTIL E…) in the actual library.

Step 10: Index Records that Have Been Updated in the Interim
In order to include records that have been updated while indexing has been running in the indexing library, activate UTIL/E/5/2 in the actual library. This copies the saved Z07H records to Z07, deleting duplicate entries. The ongoing ue_01 process in the actual library re-indexes the records stored in Z07.

13 Indexing Services

Each service is identified in the Batch Log and Batch Queue by its procedure name.

Rebuild Word Index (manage-01) Target

This service updates the Word Index of the database. It locks the ALEPH system and should only be run when the library is closed.

Update Direct Index (manage-05)

This service updates the Direct Index of the database. It locks the ALEPH system and should only be run when the library is closed.

Update Headings Index (manage-02)

This service updates the Headings Index of the database. It locks the library and should only be run when the library is closed.

Note that when you run this service in "Rebuild Entire Headings Index" mode, you must run the "Alphabetize Long Headings" (manage-17) service in order to have correct alphabetization.

Update Sort Index (manage-27)

This service updates the Sort Index of the database. It locks the library and should only be run when the library is closed.

Alphabetize Headings - Setup (manage-16)

This setup service alphabetizes the headings according to the rules for alphabetization that are kept in the `tab00.lng` table and the `tab_filing` table.

These rules create a "filing text" by which the heading is alphabetized. The headings are then alphabetized according to the first 69 characters of the filing text of each entry.

The Alphabetize Long Headings (manage-17) service alphabetizes those headings whose filing texts are longer than 69 characters.

After you run this service, always run the Alphabetize Long Headings (manage-17) service. This service locks the library and should only be run when the library is closed.

Alphabetize Long Headings (manage-17)

This service alphabetizes those headings whose filing texts are longer than 69 characters. This service locks the library and should only be run when the library is closed.

Update Short Bibliographic Records (manage-07)

This service updates the Short Bibliographic Records of the database. It locks the library and should only be run when the library is closed.

Note that if the CREATE-Z00R variable of the tab100 table in the library's tab directory is set to Y, this function also updates/creates Z00R records. The Z00R table contains separate Z00R records for each of the fields in all documents of the database. This information can be used for statistical purposes.

Update Indexes for Selected Records (manage-40)

This service writes the requested document numbers in the Z07 Oracle table, after which the records are re-indexed through the library's usual updating process (ue_01).

Build Counters for Logical Bases (manage-32)

This service builds the counters for logical bases. It should be run after building a Headings index (manage-02) if your database setup is configured for using this counter.

Update Brief Records (manage-35)

This service updates and creates brief records. It locks the ALEPH system and should only be run when the library is closed.

Create Links Between Records (manage-12)

This service creates links between records of the database. It locks the library and should only be run when the library is closed.

In order to re-build all links:

1. Empty z103 (=util a/17/1) from BIB and HOL libraries (and CRS if there is one).
2. Run p-manage-12 in ADM with parameters: delete all=Y, check old=Y.
3. Run p-manage-12 in BIB with parameters: delete all=N, check old=Y.
4. Run p-manage-12 in HOL with parameters: delete all=N, check old=Y.

14 Further Reading

The following ALEPH documents contain material not included in the Indexing chapter. They are available from the Ex Libris Documentation Center. Note that differences might appear between these documents and the current version.

How To Run Index Jobs - This document describes how to run index jobs such as Words and Headings. It touches upon such issues as turning off archive logging, number of processes, disk space and file locations, unlocking the library while a job is running, monitoring jobs, troubleshooting, and estimation runtime.

Index Building / Parallel Processing - This document explains how parallel processing can be used in ALEPH to decrease the time of indexing, within specific constraints. It describes key concepts such as cycles, job duration, disk space calculation and acquaints the reader with the batch jobs involved. It also points out when part of a specific task cannot be done in parallel.