SOFTWARE FOR DATA IMPORT

manual

product:	CBS
author:	CBS Team
date:	10 November 2010
document name:	Software for Data Import.doc
document status:	RELEASED
version:	3.8
remarks:	

Contents

Revision History

Date	Version	Author	Comments
28-01-2002	1.0	Rob Prins	Creation.
28-06-2002	2.0	Ryan Sutherland	Major update to CSNRM library and all csfn_ programs.
05-09-2002	2.1	Ryan Sutherland	Added documentation for abes_csfn_fmesh2norm and abes_csfn_localtagsnorm.
02-10-2002	2.2	Maarten Heesakkers	Updated chapter "Title storage: csfn_storenorm".
12-12-2002	2.3	Ryan Sutherland	Added –w option to csft_displayfiles.
16-12-2002	2.4	Ryan Sutherland	Added new program csfn_fcvreversenorm.
08-01-2003	2.5	Ryan Sutherland	Updated csfn_pica32norm for the new –k option, and added this “revision history” table (with as much data as possible).
09-01-2003	2.6	Paul Smit	Added syntactical validation to csfn_valnorm
27-01-2003	2.7	Ryan Sutherland	Removed sections for provider-specific programs. Converted this document to OCLC-PICA template.
17-02-2003	2.8	Ryan Sutherland	Updated csfn_fcvnorm for new sorting option “-t”.
12-07-2004	2.9	Ryan Sutherland	UNICODE project: updated with changes for utf8 processing
12-11-2004	2.10	Guido van den Heuvel	Added note on checking UTF8 in csfn_checknorm; removed some typo's
25-01-2005	2.11	Huug Peters	csfn_fixnorm added
03-02-2005	2.12	Ryan Sutherland	Moved documentation about offline tools into a separate document: this document now discusses only import software.
28-07-2005	2.13	Ryan Sutherland	Updated documentation for csfn_seqnorm
29-07-2005	2.14	Guido van den Heuvel	Added documentation for csfn_storebulk.
02-08-2005	2.15	Ryan Sutherland	Updated documentation for csfn_seqnorm and csft_rangenorm
18-01-2006	3.0	Bram van Dam	Added documentation for csfn_xsltxml2norm and csfn_xsltxmlparser
13-04-2006	3.1	Huug Peters	Added csfn_enricharticle and csfn_storearticle
18-09-2007	3.1.1	Els van Doorn	Added csfn_utf8fixnorm
28-09-2007	3.2	Ryan Sutherland	Grouped csfn_utf8fixnorm, csfn_fixnorm and csfn_checknorm into a single chapter
13-11-2008	3.3	Hetty van Zutphen	Csfn_rangenorm extended with ttltype selection
8-12-2008	3.4	Ryan Sutherland	Converted to new OCLC document template. No content changes.
12-01-2009	3.5	Ryan Sutherland	Minor changes to csfn_fixnorm and csfn_checknorm
9-10-2009	3.6	Ryan Sutherland	Added csfn_storemm
21-09-2010	3.7	Ryan Sutherland	Added csfn_sortnorm
10-11-2010	3.8	Ryan Sutherland	Added notes about Batch Log and APC Log entries
20-12-2010	3.9	Huug Peters	Added csfn_perlnorm
08-02-2019	3.7	Paul Smit	Added chapter "Removing punctuation with an APT table"

References

Developers are referred to documentation describing the following software libraries:

- FCV
- CCV
- VAL
- PCCCAT
- CSNRM

O&S users are asked to see the document “Usage of csfn_importconvert.doc”, which details the proper usage of the shell script.

1 Introduction

1.1 Scope of this Document

This document describes how to use the software that has been developed to import data into a Central System. This document does not describe software requirements, design, or testing details.

1.2 Intended Audience

This document is intended for the users of the import software, both internal and external. In addition, developers may find this document useful to become familiar with this software package.

2 Normalized files

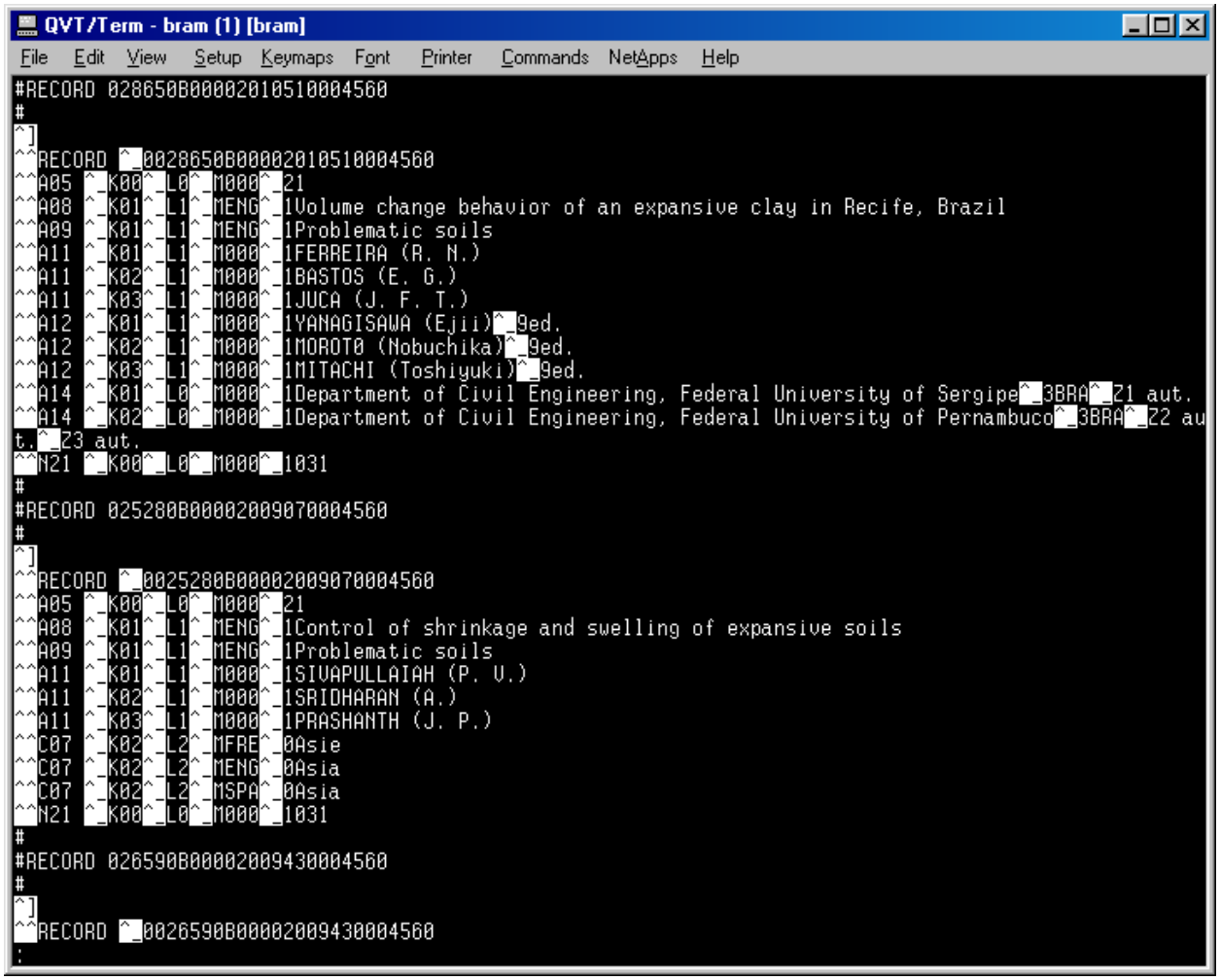
The normalized file format is a means for dealing with title data outside the Sybase databases, and is especially designed for the purpose of converting data. The idea behind this is to simplify the converting/updating process, as most parts of it will be available as a set of standard tools. This way, only a small program has to be written for each type of input file, and the tasks of converting character sets and tag definitions can be left to existing software.

2.1 The normalized title format

- A normalized title file consists of zero or more records, consisting of one or more tags, consisting of one or more subfields.
- All data is on lines of text, terminated by a single LF character (ASCII 10); this may be omitted on the last line.
- Each record's beginning is marked by a line with a single record separator character on it (ASCII 29). All data before the first record separator (ASCII 29) is ignored.
- The maximum size of a record is 32 kB. Larger records are considered empty by the tools, as they won't fit in their memory buffer. These overflowing records can be picked out by using csfn_checknorm.
- Records can be empty. Empty records are represented by two record separators with nothing in between but comment, or just a LF (ASCII 10). In the filter tools, empty records are written on standard output (if applicable) but nothing further happens.
- If the record is not empty, the first character on the next line should be a tag separator character (ASCII 30), followed by a tag name (with or without occurrence), a space and a subfield separator (ASCII 31). The rest of the record is defined by repeated tag and subfield separators and the text between them. The record is considered complete at end of file, or if a new record separator is found.
- Only printable ASCII characters (greater than 31) are allowed as subfield data. They can be filtered out of normalized title files with csfn_checknorm.
- Tag names consist of one or more letters, numbers and/or @'s, and nothing else.
- The occurrence of a tag is defined by placing a slash (/) and one or two digits directly behind the tag name. Asterisks (*) can be used as wildcards, e.g. "0123/0*". If no occurrence is given, it is supposed to be 00.
- A subfield is defined by the subfield separator (ASCII 31), followed by a one letter or digit identifier.
- Tags should always contain one or more subfields.
- Subfields can be empty.
- It is possible to spread a tag or subfield over several lines, or put several tags on a single line; however, for the sake of easy manual analysis a file this is not recommended.
- A normalized title file may contain lines beginning with a hash (#). These lines are considered comments, and completely ignored by all of the tools. Empty lines will also be ignored.

- The normalized title definition does not specify a mandatory character set or tag structure -- one of its purposes is to make it easier to convert from one character set/tag format to the other.

An example of a normalized title file:

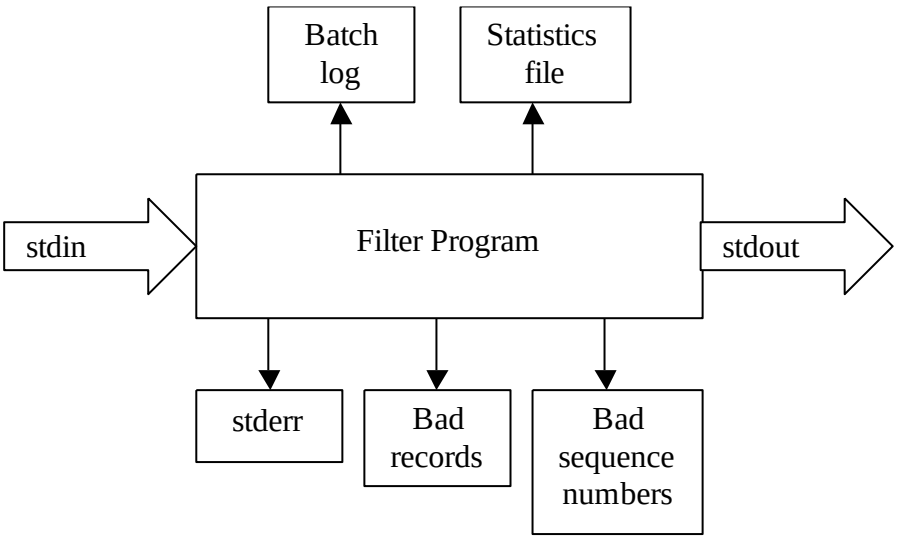


3 General Tool Design

The normalized file tools are a set of Unix filters, reading from standard input sending output in six possible directions:

1. Successfully processed output is written to stdout.
2. Titles (records) that are not successfully processed are written to an error file.
3. The sequence numbers corresponding to the titles that are not successfully processed are written to a file. The intention is that this file of sequence numbers will be used to repair and reprocess the titles that failed processing.
4. System and program errors are written to stderr.
5. Program statistics are written to the batch log. This behaviour is optional.
6. Program statistics are written to a text file, commonly used for the gathering of management statistics. This behaviour is optional.

The following diagram illustrates the general structure of each of the filter programs.



3.1 Batch Log Entries

All import software that has been developed according to this design makes the following entries in the table gen_batch_log during an 'actual' run:

1. Run started (counter_id=736)
2. Number of records read (counter_id=720)
3. Number of records written (counter_id=700)
4. Number of empty records found (counter_id=716)
5. Number of records that caused errors (counter_id=728)
6. Run stopped (counter_id=737)

Note that the record counters make no distinction among records from different record-levels (ie: main, local, copy and social).

3.2 APC (Automatic Process Control) Log

All import software that has been developed according to this design makes an entry in the gen_apc_log during an 'actual' run that contains data about the job, including the start and stop times.

4 The Import Process

Typically, a conversion of any data into a Sybase database would take the following steps:

1. a normalization filter, designed for the specific input format, normalizes titles;
2. **csfn_ccvnorm** converts the titles to the desired character set;
3. **csfn_fixnorm** removes empty subfields and empty tags, removes leading and trailing blanks;
4. **csfn_checknorm** checks the titles for any record-format errors;
5. **csfn_fcvnorm** converts the titles to comply to the Pica+ tag definitions;
6. **csfn_utf8norm** converts the record to utf8 format for database insertion;
7. **csfn_valnorm** checks the titles for valid syntax;
8. **csfn_storenorm** writes the converted titles into the Sybase database;

The programs can be chained together in a UNIX pipeline, or they can be run individually, as the situation demands.

The precise set of actions required for data import depends, of course, upon the data itself. Any of the steps can be omitted if they are not necessary. Several general guidelines do apply to the sequence of steps; these guidelines only apply when the named steps are required:

1. the normalization step must occur first
2. character conversion, record fixing and format checking (csfn_ccvnorm, csfn_fixnorm and csfn_checknorm) must occur before format conversion (csfn_fcvnorm)
3. character conversion, record fixing and format checking (csfn_ccvnorm, csfn_fixnorm and csfn_checknorm) tend to occur in that order
4. utf8 conversion (csfn_utf8norm) must occur after format conversion (csfn_fcvnorm) and before both validation and storage (csfn_valnorm and csfn_storenorm)
5. validation (csfn_valnorm) must occur after format conversion (csfn_fcvnorm) and immediately before storage (csfn_storenorm)
6. storage (csfn_storenorm) is the last step in the process

5 Normalization Filters

5.1 Marc Normalization: csfn_marc2norm

The program csfn_marc2norm is a filter for normalizing Marc-format titles. In the output of csfn_marc2norm, titles that couldn't be properly normalized are written to the error file and are replaced by empty records in the stdout stream.

```
*** Usage for csfn_marc2norm:

This program reads a file with marc-format titles from stdin,
normalizes the title, and writes the normalized titles to stdout.
System- and program-level errors are written to stderr.

-g <Segmentation Type>
                N - None(default)
                U - USML
                C - CCNPS
-t <offset>     Offset of input file [0..99]
-n <normalization format>
                USX  - usmarc-exchange(default)
                UNX  - unimarc-exchange
                UKX  - ukmarc
                ABES - abes marc
                CLC  - clc marc
-o <origin>     Identifies the origin of the input file.
                Used as a key to get the master record. Maximum length is 9.
-i <filename>   The filename of the original import file. Maximum length is
                65.
-e <errorfile>  Titles that fail will be written to this file.
-b <badsequencefile> Sequence numbers of titles that fail will be written
                to this file.
-r <reprocessfile> The sequence numbers to be reprocessed will be read
                from this file.
-s <statisticsfile> Statistics of program performance will be written
                to this file.
-a              Actual run: use log files and write to database.
-p              Print to output stream only records that process
                successfully.
```

An example usage:

```
$ csfn_marc2norm -g N –a -o DDB -n USX -t 0 -i stdinfile -e errorfile \
      –b badsequencefile -s statisticsfile 2>> stderrfile < stdinfile \
      > stdoutfile
```

5.2 Netfirst Normalization: csfn_netfirst2norm

The program csfn_netfirst2norm is a filter for normalizing netfirst-format titles. In the output of csfn_netfirst2norm, titles that couldn't be properly normalized are written to the error file and are replaced by empty records in the stdout stream.

```
usage: csfn_netfirst2norm [-a -o origin -i filename] [-e errorfile]
                [-b badsequencefile] [-s statisticsfile] [-h]

This program reads a file with netfirst-format titles from stdin,
normalizes the title, and writes the normalized titles to stdout.
System- and program-level errors are written to stderr.

-a              Actual Run: use log files and write to database.
-o origin       Identifies the origin of the input file.
                Used as a key to get the master record.
-i filename     The filename of the original import file.
                Do not include the entire path.
-e errorfile    Titles that fail will be written to this file.
                Default: "error_nrm".
-b badsequencefile Sequence numbers of titles that fail will be written
                to this file.  Default: "sequence_nrm".
-s statisticsfile Statistics of program performance will be written
                to this file.
-h              Print this screen and exit the program.
```

An example usage:

```
$ csfn_netfirst2norm -a -o IDSTRING -i stdinfile -e errorfile -b badsequencefile \
      -s statisticsfile 2>> stderrfile < stdinfile > stdoutfile
```

5.3 Pica3 Normalization: csfn_pica32norm

The program csfn_pica32norm is a filter for normalizing pica3-format titles. In the output of csfn_pica32norm, titles that couldn't be properly normalized are written to the error file and are replaced by empty records in the stdout stream.

This program accepts several different record syntaxes, and the –k flag can be used to select which syntax to use. The RGN record syntax (this is the default) is as follows: a record STARTS with a record separator and each tag is on its own line. All lines end with ASCII 10, which is the Linefeed character. For example:

```
9 10
'tttt contents' 10
'tttt contents' 10
'tttt contents' 10
```

The BIBSERV syntax is as follows: ASCII 10 functions as the record separator, but this time it signals the END of the record. ASCII 30 is the tag separator. For example:

```
'tttt contents' 30 'tttt contents' 30 'tttt contents' 30 'tttt contents' 30 10
```

The SWETS syntax is as follows: ASCII 13 functions as both the tag separator and the record separator, for example:

```
'tttt contents' 13 'tttt contents' 13 13
```

The –u flag should be used if subfield separators are desired in the output stream. Otherwise, no subfield separators are generated.

The –m flag should be used if special MPSP tags are present and require special conversion.

```
*** Usage for csfn_pica32norm:

This program reads a file with pica3-format titles from stdin,
normalizes the title, and writes the normalized titles to stdout.
System- and program-level errors are written to stderr.

-k <syntax>      Specifies record syntax:
                     RGN: RGN-specific record format (default)
                     BIBSERV: BIBSERV-specific record format
                     SWETS: SWETS-specific record format
-m               Convert MPSP tags.
-u               Insert subfield separators.
-y               (Obsolete) Specifies BIBSERV record syntax.

Please see CSNRM library documentation for extended usage details.
```

An example usage:

```
$ csfn_pica32norm –k SWETS -a -o IDSTRING -i stdinfile -e errorfile \
      –b badsequencefile -s statisticsfile 2>> stderrfile < stdinfile > stdoutfile
```

5.4 Mab Normalization: csfn_mab2norm

The program csfn_mab2norm is a filter for normalizing MAB-format titles. In the output of csfn_mab2norm, titles that couldn't be properly normalized are written to the error file and are replaced by empty records in the stdout stream.

```
usage for csfn_mab2norm:

This program reads a file with mab-format titles from stdin,
normalizes the title, and writes the normalized titles to stdout.
System- and program-level errors are written to stderr.

-o <origin>      Identifies the origin of the input file.
                 Used as a key to get the master record. Maximum length is 9.
-i <filename>    The filename of the original import file. Maximum length is
                 65.
-e <errorfile>   Titles that fail will be written to this file.
-b <badsequencefile> Sequence numbers of titles that fail will be written
                 to this file.
-r <reprocessfile> The sequence numbers to be reprocessed will be read
                 from this file.
-s <statisticsfile> Statistics of program performance will be written
                 to this file.
-a               Actual run: use log files and write to database.
-p               Print to output stream only records that process
                 successfully.
```

An example usage:

```
$ csfn_mab2norm –a –o BSZ –i stdinfile –e errorfile –b badsequencefile \
      -s statisticsfile 2>> stderrfile < stdinfile > stdoutfile
```

5.5 Text normalization: csfn_perlnorm

The program csfn_perlnorm is a filter for normalizing text files. Such files may arrive from various sources and will arrive in many different formats. In order to have this filter make something useful out of the input data a few demands on the structure of the input data must be met:

1) The beginning of each record must be clearly defined in the input.
2) Each record starts at new line.

```
Usage:
csfn_perlnorm.pl hazb:e:i:o:pr:s:x:y:t:l:m:u:
Options for compatibility, with no effect:
a z i o r t
Options common to normalized data processing
h       Print this information
b<file> Write title sequence numbers of records with processing errors to
<file>
e<file> Write input records with processing errors to <file>
p       Write empty outputrecords for input records with processing errors to
STDOUT
x<nr>   Start processing at title sequence number <nr>
y<nr>   Stop processing at title sequence number <nr> (inclusive)
Options specific for csfn_perlnorm.pl:
l<rexp> beginning of record matches regular expression <rexp>
m{N|P|L=<file>}
        Input is Normalized | Plain | Labeled
        use <file> to convert labeled input to normalized output
u<file> call perl subroutine repairRecord from this module
```

An example usage:

```
$ csfn_perlnorm –a -o BSZ -i stdinfile -e errorfile –b badsequencefile \
      -s statisticsfile –l'^Record' –mL=labelsfile 2>> stderrfile < stdinfile
       > stdoutfile
```

5.5.1 Different modes for different sorts of input data

The filter supports different modes, for various types of input data:

L=<file>	Labelled mode	tags are labelled lines, <file> contains regular expressions for each label and its replacement. E.g.: "Title:\s*"="^^021A ^_"
N	Normalized mode	The input record is already in normalized mode, this mode is typically used when already normalized records need some repair.
P	Plain mode	Each line is prefixed with "^^*nnn* ^_0", *nnn* being the line number since the last start of a new record

The "^^" and "^_" are the regular begin of tag and begin of subfield markers for normalized data.

5.5.2 The repair module

The repair module allows the user to inject a repair subroutine into the process. Possible forms of use of such a subroutine are:

- Combine continuation lines for labelled or plain input data.
- Repair data from multiple input lines

Here is a brief example of this subroutine, which must always be named repairRecord. The example can be found in the file csfn_perlrepair_example.pm

```
sub repairRecord($$)
#
# The subroutine is invoked with two array references.
# The first one, $record, refers to the title record to be repaired
# The second one, $errorMessages is an array of error messages to which
# the error messages from this subroutine should be appended
#
{
    my (
            $record,
            $errorMessages) = (@_);
    #
    # look at every indvidual line in the record
    #
    for my $tag (@$record) {
        if ($tag =~ /some string/) {
            push @$errorMessages, "some message to append";
        }
    }
}
```

6 XML Normalization and XSLT based conversions

6.1 Introduction

There are two programs to convert a XML formatted title to a normalized title:
csfn_xsltxml2norm and csfn_xsltxmlparser

csfn_xsltxmlparser processes an entire file with titles at once.
csfn_xsltxml2norm processes each title separately.

csfn_xsltxmlparser is easy to use but misses some flexibility. csfn_xsltxml2norm provides all normal csfn functionality but is slightly harder to use.

csfn_xsltxmlparser can be used for any XML to "whatever text file conversion" using an appropriate XSLT-style sheet. This can for example be from XML to XML, XML to HTML, XML to C or XML to plain text.

Both programs read the XSLT style-sheet and the XML-titles and apply the style sheet to the titles according to the rules as defined in the XSLT-standard.

The result is send to stdout.

6.2 Differences between csfn_xsltxml2norm and csfn_xsltxmlparser

The most important differences between the programs are:

1) csfn_xsltxml2norm processes one record at a time, while csfn_xsltxmlparser processes the entire file at once.
2) csfn_xsltxml2norm requires to know the tag that defines the title. For example: <record> in MarcXML.
3) csfn_xsltxml2norm works with the normal common csfn-program parameters. Therefore you can select which titles to process and use all other csfn-functionality. csfn_xsltxmlparser does not use these parameters. Therefore you can not use the normal flexibility with this program. The parameters are only accepted for script-compatibility reasons.
4) csfn_xsltxml2norm requires the XSLT-style sheet to be useable for a selection of the xml-structure. Therefore it does not always work without modification of the style sheet.
5) csfn_xsltxmlparser can also be used for any other XML conversion using a XSLT style sheet.

Under normal circumstances it is advised to use csfn_xsltxmlparser unless you require common csfn functionality.

6.3 How to create a XSLT template for normalized format?

This is almost the same as writing an XSLT template for any XML to text conversion. See the XSLT standard and/or a good tutorial if you need to know more about creating a XSLT-style sheet.

The only problem with creating a XSLT for normalized format is that the control characters for record-, tag- and subfield-separation (ASCII 29,30 and 31) are not valid XML-characters and can therefore not be used in the XSLT-style sheet.

Therefore we use the following tokens for these characters. These tokens will be replaced by the corresponding control character by the program.

"PICANORMRECMARKER"	marks the start of a new record:	ascii 29
"PICANORMTAGMARKER"	marks the start of a new tag:	ascii 30
"PICANORMSUBMARKER"	marks the start of a new subfield:	ascii 31

6.4 How to use the programs?

Run the program with the appropriate parameters. To tell the program which style sheet to use you need to tell the program either a filename (-f) or a tablekey (-k) for the style sheet. Don't use both. Normally the table key option is to be used.

Under most circumstances it is advised to use csfn_xsltxmlparser, unless you want to explicitly use the common csfn-programs functionality.

Due to possible simplifications of the style sheet it can be generally advised to run csfn_fixnorm over the result to fix potential errors.

6.4.1 csfn_xsltxmlparser

Usage: csfn_xsltxmlparser [-h] -f<> -t<> [-w] [-n]
Options:

-h	This is help screen
-f filename	File with style sheet
-k tablekey	Table key for style sheet

Use either -k or -f for style sheet

-w	Ignore xmlns in titles (only use this Workaround if it does not work normally)
-n	Result is in normalized format

The normal common parameters for a normalized program are also accepted. However, because they are meaningless for this program they are ignored. They are only accepted for script compatibility reasons.

Explanation of program specific parameters:
-w: This option is used to tell the program to ignore namespace references and should only be used when it does not work without this option. The problem arises due to a broken link used in many xml-files.
-n: This option is used to tell the program that the result is in normalized format. The program replaces the "PICANORM*MARKER" tokens by their corresponding ASCII characters. The program will also remove any xml-header in the result.

With the absence of the -n option the program can be used for any XML-conversions with XSLT-style sheets. For example: from Dublin Core XML to MARCXML. With the -n option you can convert to normalized format. For example form MARCXML to Marc21. Running this program twice in this example gives us a conversion from Dublin core to Marc21.

6.4.2 csfn_xsltxml2norm

Usage: csfn_xsltxml2norm -f -n -c [-h] [-o] [-i] [-e] [-b] [-r] [-s] [-a] [-z] [-p]
Options:

-h	This help screen
-f <xslt file>	Name of file with xslt style sheet
-k tablekey	Table key for style sheet

Use either -k or -f for style sheet

-n <name>	Name of tag that defines a record
-c	Replace single '&' in URL's by &

Explanation of program specific parameters:
-n: This option tells the program with xml-tag defines a title. For example the <record> tag in MarcXML. This tag is used to split the file into its separate titles.
-c: This option is used to tell the program to correct a certain bug in the data and should become obsolete. If this option is given all single '&' characters are replaced by their correct xml-representation: '&'.

The following options are the common normalization program options. csfn_xsltxml2norm normally accepts and work with these options. csfn_xsltxmlparser accepts and ignores these options.

-a	Actual run: use log files, perform other serious actions.
-b <badfile>	Sequence numbers of records that fail will be written to this file.
-e <errorfile>	Records that fail will be written to this file.
-i <filename>	The filename of the original import file. Maximum length is 256.
-o <origin>	Identifies the origin of the input file. Used as a key to get the master record. Maximum length is 9.
-p	Print to output stream only records that process successfully.
-r	Reserved.
-s <statsfile>	Statistics of program performance will be written to this file.
-x	Title sequence number to start at (range filtering)
-y	Title sequence number to end at (range filtering)
-z	Lean-and-mean mode: do not update logs or other tables (Only useful when -a is used)

6.5 Example usages

Here I will give some examples of using the programs. See next paragraph for an example of a style sheet. It is advised to run csfn_fixnorm after a normalization run of one of the programs. The programs do not check whether the style sheet delivers correct normalized results.

example 1:

```
csfn_xsltxmlparser -n -k XSLT#STYLESHEET#MARCXML2MARC21 <xmldatain
>normdataout
```

During this run the program reads the data in xmldatain, applies the style sheet "XSLT#STYLESHEET#MARCXML2MARC21" to it, replaces the norm tag, subfield and record tokens and writes the result to normdataout.

example 2:

```
csfn_xsltxmlparser -f csfn_xsltxmlparser_DC2MARCXML.xsl.example <xmldata
```

During this run the program reads the data in xmldata and the style sheet in csfn_xsltxmlparser_DC2MARCXML.xsl.example. It applies the style sheet to the data. The result is written to stdout.

The source in this example was in Dublin core format, the result is in MarcXML format.

example 3:

```
csfn_xsltxmlparser -f csfn_xsltxmlparser_DC2MARCXML.xsl.example <xmldata |
csfn_xsltxmlparser -n -k XSLT#STYLESHEET#MARCXML2MARC21 >normdataout
```

This train combines example 1 and 2. The source data was in Dublin core format. After two runs of the program the result is in MARC21 format. This saves us the time of writing a third style sheet.

example 4:

```
csfn_xsltxml2norm -e err -n record -k XSLT#STYLESHEET#MARCXML2MARC21
<marcxmldatain >normdataout
```

During this run the program reads the data in marcxmldatain. It looks for the first <record> tag, applies the style sheet to the record and writes the result to normdataout if all went well and to err if it did not. Then it looks at the next <record> tag and applies the style sheet etc. till all <record> tags have been processed in this manner.

6.6 Example: MarcXML to MARC21 XSLT style sheet

The following example of an XSLT style sheet converts MarcXML titles into MARC21 titles.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version = '1.0'
     xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>

<!-- This template contains the tokens PICANORMRECMARKER,
     PICANORMTAGMARKER and PICANORMSUBMARKER. These are used
     by the software to place ascii 29-31 which can not be
     coded directly -->

<xsl:template match="/">
     <xsl:apply-templates select="//record"/>
</xsl:template>

<xsl:template match="collection">
     <xsl:apply-templates select="//record"/>
</xsl:template>


<xsl:template match="record">
     <xsl:text>PICANORMRECMARKER</xsl:text>
     <xsl:apply-templates select="./leader"/>
     <xsl:apply-templates select="./controlfield"/>
     <xsl:apply-templates select="./datafield"/>
</xsl:template>  <!-- record -->

<xsl:template match="leader">
    <xsl:text>PICANORMTAGMARKER000 PICANORMSUBMARKER0</xsl:text>
    <xsl:value-of select="."/>
</xsl:template> <!-- leader -->

<xsl:template match="controlfield">
    <xsl:text>PICANORMTAGMARKER</xsl:text>
    <xsl:value-of select="@tag"/>
    <xsl:text> PICANORMSUBMARKER0</xsl:text>

    <xsl:value-of select="."/>
</xsl:template> <!-- controlfield -->

<xsl:template match="datafield">
         <xsl:text>PICANORMTAGMARKER</xsl:text>
         <xsl:value-of select="@tag"/>
         <xsl:text> </xsl:text>

        <!-- Parse ind* attributes -->
        <xsl:for-each select="@node()">
                <xsl:variable name="ind" select="."/>
        <xsl:if test="position()!=1">       <!-- not attr "tag" -->
           <xsl:choose>
                <xsl:when test="$ind = ''">
                           <xsl:text> </xsl:text>
                </xsl:when>
                <xsl:otherwise>
                           <xsl:value-of select="$ind"/>
                </xsl:otherwise>
           </xsl:choose>
```

```
        </xsl:if>
        </xsl:for-each>

         <xsl:apply-templates select="./subfield"/>
</xsl:template> <!-- datafield -->

<xsl:template match="subfield">
    <xsl:text>PICANORMSUBMARKER</xsl:text>
    <xsl:value-of select="@code"/>
    <xsl:value-of select="."/>
</xsl:template> <!-- subfield -->
</xsl:stylesheet>
```

7 Character Conversion: csfn_ccvnorm

The program csfn_ccvnorm is a filter for converting normalized title data from one character set to another. In the output of csfn_ccvnorm, titles that couldn't be converted are written to the error file and are replaced by empty records in the stdout stream.

Note that the table key specified by the –k option is necessary for this program to run. If no valid table entry is found, this program will not process any titles and will return an error code.

```
*** Usage for csfn_ccvnorm:

This program reads a file with titles from stdin, converts the titles
with a CCV table and writes the converted titles to stdout.
Titles that produce errors are written to the error file.
System- and program-level errors are written to stderr.

-k <key>          Key of the CCV table to use.
-o <origin>       Identifies the origin of the input file.
                  Used as a key to get the master record. Maximum length is 9.
-i <filename>     The filename of the original import file. Maximum length is
                  65.
-e <errorfile>    Titles that fail will be written to this file.
-b <badsequencefile> Sequence numbers of titles that fail will be written
                  to this file.
-r <reprocessfile> The sequence numbers to be reprocessed will be read
                  from this file.
-s <statisticsfile> Statistics of program performance will be written
                  to this file.
-a                Actual run: use log files and write to database.
-p                Print to output stream only records that process
                  successfully.
```

An example usage:

```
$ csfn_ccvnorm -k CCVDEF#acad#utf8 -e ccverror.txt -b badsequencefile \
      -s statisticsfile < stdinfile > stdoutfile 2> stderrfile
```

8 Sorting Records: csfn_sortnorm

The program csfn_sortnorm is used for sorting normalized records by tag name. Currently only 'Pica+' and 'alphabetical'sorting are supported.

```
*** Usage csfn_sortnorm [-t]
This tool sorts normalized records based on tag codes.

Options:
-t <method>      Sorting method.  Options:
                    1.  PICA+   (default)
                    2.  ALPHA   (alphabetical)
-h               This help screen

The following options are common to all normalization programs:

-a               Actual run: use log files, perform other serious actions.
-b <badfile>     Sequence numbers of records that fail will be written
                 to this file.
-e <errorfile>   Records that fail will be written to this file.
-i <filename>    The filename of the original import file. Maximum length is 256.
-o <origin>      Identifies the origin of the input file.
                 Used as a key to get the master record. Maximum length is 9.
-p               Print to output stream only records that process successfully.
-r               Reserved.
-s <statsfile>   Statistics of program performance will be written
                 to this file.
-x               Title sequence number to start at (range filtering)
-y               Title sequence number to end at (range filtering)
-z               Lean-and-mean mode: do not update logs or other tables
                 (Only useful when -a is used)
```

An example usage:

```
$ csfn_sortnorm –t ALPHA -e normerror.txt -b badsequencefile \
      –s statisticsfile < stdinfile > stdoutfile 2> stderrfile
```

9 Checking and fixing records before format conversion

There are now several tools available to check and fix records before sending them to format conversion (csfn_fcvnorm). Use of these tools is not mandatory but is recommended in order to prevent as many processing errors as possible during format conversion.

9.1 Fix utf8: csfn_utf8fixnorm

This program is intended for use on records already encoded in UTF8 and will fix the following problems, if present:

- Converts NCR, the "Numeric Character References," into UTF8
- Converts the given UTF8 into the normalized format of UTF8 used in the Pica database
- Recovers incomplete diacritics

In the output of csfn_utf8fixnorm, titles that couldn't be converted are written to the error file and are replaced by empty records in the stdout stream.

Strictly speaking, this program can be used either before or after csfn_fcvnorm, but it is recommended that it be used before attempting any format conversion with csfn_fcvnorm. Furthermore, it is recommended to use this program prior to csfn_checknorm as that program checks for invalid UTF8 encoding.

```
*** Usage for csfn_utf8fixnorm:

This program reads a file with titles from stdin, converts the titles
with the PU library and writes the converted titles to stdout.
Titles that produce errors are written to the error file.

System- and program-level errors are written to stderr\
-h               Display this help screen
-v               Print verbose debugging information

The following options are common to all normalization programs:
-a               Actual run: use log files, perform other serious actions.
-b <badfile>     Sequence numbers of records that fail will be written
                 to this file.
-e <errorfile>   Records that fail will be written to this file.
-i <filename>    The filename of the original import file. Maximum length is
256.
-o <origin>      Identifies the origin of the input file.
                 Used as a key to get the master record. Maximum length is 9.
-p               Print to output stream only records that process
successfully.
-r               Reserved.
-s <statsfile>   Statistics of program performance will be written
                 to this file.
-x               Title sequence number to start at (range filtering)
-y               Title sequence number to end at (range filtering)
-z               Lean-and-mean mode: do not update logs or other tables
                 (Only useful when -a is used)
```

An example usage:

```
$ csfn_utf8fixnorm -e utf8fixerror.txt -b badsequencefile \
      -s statisticsfile < stdinfile > stdoutfile 2> stderrfile
```

9.2 Fixing titles: csfn_fixnorm

The program csfn_fixnorm scans input for non-fatal input errors. The program simply passes such input as comment (line starting with '#') to stdout. No messages or errors arise.

The following anomalies in records are fixed:

- Leading and trailing blanks are deleted from variable-length subfields
- Empty subfields are removed and reported in comment.
- Empty tags are removed and reported in comment.
- Blanks immediately following subfield mark are removed, first non-blank is made subfield type.

```
*** Usage: csfn_fixnorm [-q mm*][-m][-h][-o][-i][-e][-b][-r][-s][-a][-z][-p]

This program reads a file with titles from stdin,
removes leading and trailing blanks from subfields
removes empty subfields and removes tags without any subfield,
System- and program-level errors are written to stderr.

Options:
-q mm*          Tag mask to point out tags NOT to be fixed.
                -q option can be repeated to enter multiple masks
                mask-tag comparison is done for just the length of mask
                m =[0-9A-Za-z*]  An asterisk matches any character
                Examples:
                "-q 00"  will leave unchanged tag 0001
                "-q 0*0" will leave unchanged tag 0001
                "-q 0*0" will leave unchanged tag 0101
                "-q 0*0" will fix tag 01
-m <opt>        Modification to make to the records.  Available options:
                "indicators" all non-alphanumeric tag indicators will be
                             replaced by spaces
-h              This help screen
```

The program has a possibility to pass multiple tag masks using one or more –q options. This allows the user to have a mix of fixed-format tags that remain unchanged in the stream while other tags are being fixed. The program has no limit to the number of –q options on the command line.

csfn_fixnorm can be used on any type of normalized records and can be used either before or after csfn_fcvnorm in the import process. It is recommended, however, to use it before csfn_fcvnorm to prevent as many errors as possible.

The –m "modifications" option currently takes one option, that to modify the tag indicators: any indicators that are not alphanumeric will be changes into space characters, and any lowercase indicators will be changed to uppercase.

It is recommended to use csfn_fixnorm before csfn_checknorm.

9.3 Checking title syntax: csfn_checknorm

The program csfn_checknorm searches the input file for corrupt title records, and replaces any corrupt titles with empty ones. While this program may be run following normalization, character conversion or format conversion, it is recommended to run this program as the last step before format conversion.

If an error file name is given, the corrupt records are written to this file, along with a description of the found error.

The following corruptions are detected:

- record too large for memory buffer (32 kB);
- any other character than a tag separator at the beginning of the record;
- zero byte(s) in a title;
- any control characters in a title, apart from LF and tag/subfield separator;
- tags with no subfields;
- incorrect tag names;
- missing spaces between tag name and first subfield;
- incorrect subfield codes – note the use of the –m option to allow for the expanded set of non-alphanumeric subfield codes;
- unexpected truncation of a record;
- invalid UTF8 in a record.

```
*** Usage for csfn_checknorm:

This program reads a file with titles from stdin, checks the titles
for any syntax errors and writes the valid titles to stdout.
Titles that contain syntax errors are written to the errorfile.
System- and program-level errors are written to stderr.

-t <>            Type of input records.  Options:
                    1.  PICA+   (default)
                    2.  PICA3
                    3.  MARC
                    4.  STRX
                    5.  MAB2
-u               Input data is encoded with the utf8 character set
-m               Allow expanded set of non-alphanumeric subfield types
-h               This help screen.
```

An example usage:

```
$ csfn_checknorm -m -e chkerror.txt -b badsequencefile -s statisticsfile \
      2>> stderrfile < stdinfile > stdoutfile
```

10 Format Conversion: csfn_fcvnorm

The program csfn_fcvnorm is a filter for converting normalized title data from one format to another. In the output of csfn_fcvnorm, titles that couldn't be converted are written to the error file and are replaced by empty records in the stdout stream. Titles that are successfully converted are output in sorted by tag order.

Note that the table key specified by the –k option is necessary for this program to run. If no valid table entry is found, this program will not process any titles and will return an error code.

Also note that there are options for the sorting method, but these have been made obsolete by the creation of the csfn_sortnorm application. It is preferable to use the –t option over the –n option. If no sorting method is specified then the default method is the standard PicaPlus sorting method.

If the converter table finds any errors while processing a record, the error can be communicated to the application by placing the string “fatal” at the start of the return string. If the string “fatal” is not found, then an error will still be reported but without any descriptive details about the nature of the error.

```
*** Usage for csfn_fcvnorm:

This program reads a file with titles from stdin, converts the titles
with an FCV table and writes the converted titles to stdout.
Titles that produce errors are written to the error file.
System- and program-level errors are written to stderr.

NOTE that this program does not consider the value of the Unicode Master
Switch because the user must indicate the character set of the input data.

-k <key>          Key of the FCV table to use.
-t <>             [Obsolete: use csfn_sortnorm] Sorting method.  Options:
                     1.  PICA+   (default)
                     2.  ALPHA   (alphabetical)
                     3.  NONE
-u                Input data is encoded with the utf8 character set
-n                [Obsolete]  Indicates output records will NOT be
                  Pica+ records. Equivalent to "-t PICA+"

The following options are common to all normalization programs:

-o <origin>       Identifies the origin of the input file.
                  Used as a key to get the master record. Maximum length is 9.
-i <filename>     The filename of the original import file. Maximum length is
                  256.
-e <errorfile>    Records that fail will be written to this file.
-b <badsequencefile> Sequence numbers of records that fail will be written
                  to this file.
-r <reprocessfile> The sequence numbers to be reprocessed will be read
                  from this file.
-s <statisticsfile> Statistics of program performance will be written
                  to this file.
-a                Actual run: use log files, perform other serious actions.
-p                Print to output stream only records that process
```

```
                    successfully.
```

An example usage:

```
$ csfn_fcvnorm -k FCVDEF#pascal#pica+#full -e fcverror.txt -b sequence.txt \
      -s statistics.txt < stdinfile > stdoutfile 2> stderrfile
```

11 Reversible Format Conversion: csfn_fcvreversenorm

The program csfn_fcvreversenorm is a filter for converting normalized title data from one format to another. Unlike csfn_fcvnorm, however, this program has been designed specifically for the processing of reversible syntaxes. In the output of csfn_fcvreversenorm, titles that couldn't be converted are written to the error file and are replaced by empty records in the stdout stream. Titles that are successfully converted are output in sorted by tag order.

Note that the table key specified by the –k option is necessary for this program to run. If no valid table entry is found, this program will not process any titles and will return an error code.

```
*** Usage for csfn_fcvreversenorm:

This program reads records from stdin and converts
them from cataloguing format to internal format.
Successfully converted records are written to stdout.
Records that produce errors are written to the error file.
System- and program-level errors are written to stderr.

NOTE that this program does not consider the value of the Unicode Master
Switch because the user must indicate the character set of the input data.

-k <key>          Key of the FCV table to use.
-u                Input data is encoded with the utf8 character set

The following options are common to all normalization programs:

-o <origin>       Identifies the origin of the input file.
                  Used as a key to get the master record. Maximum length is
                  9.
-i <filename>     The filename of the original import file. Maximum length
                  is 256.
-e <errorfile>    Records that fail will be written to this file.
-b <badsequencefile> Sequence numbers of records that fail will be written
                  to this file.
-r <reprocessfile> The sequence numbers to be reprocessed will be read
                  from this file.
-s <statisticsfile> Statistics of program performance will be written
                  to this file.
-a                Actual run: use log files, perform other serious actions.
-p                Print to output stream only records that process
                  successfully.
```

An example usage:

```
$ csfn_fcvreversenorm -k FCV#pica#pica3#title -e error.txt -b sequence.txt \
      -s statistics.txt < stdinfile > stdoutfile 2> stderrfile
```

12 Removing punctuation with an APT table

MARC 21 databases are infamous for putting punctuation inside the subfields of their records, for example:

```
245 03$aLe Bureau$h[filmstrip] =$bLa Oficina
```

In this example the data inside subfield $h is '`filmstrip`' all the rest is punctuation.

Traditionally, the FCV table would remove such punctuation with FCV statements like:

```
strip(".,:;/= \"")
```

This has several disadvantages:

- it is too crude: it also removes characters that aren't punctuation characters for the pair of subfields
- it only removes punctuation from the end of subfields, even though some punctuation resides at the start of a subfield

- it overburdens the FCV table

CBS has a table specially designed to generate the right punctuation between subfields: the autopunctuation table (APT).
There are 2 flavors of APT tables:

- presentation-APT: used in (labelled) presentations to turn subfields into a string.
    - punctuations mentioned in this APT do not include subfield code
- conversion-APT: used when converting Pica+ to MARC 21
    - punctuations mentioned in this APT normally include subfield code

The import filter `csfn_apt_remove` uses the conversion-APT to remove punctuation from Pica+ subfields that have just been converted from a MARC 21 record.
As a result, `csfn_apt_remove` must be called after `csfn_fcvnorm` has converted to Pica+

For example to remove both correct and incorrect punctuation:

```
csfn_apt_remove -k autopunctuation#M21X#title
```

It's possible to only remove punctuation that is 100% correct:

```
csfn_apt_remove -k autopunctuation#M21X#title -q'remove_incorrect_punct=no'
```

For example, input record of `csfn_apt_remove`:

```
031T ^_S10$Franco$h[filmstrip] $Ca biography /$cPaul Preston.
032I $S##$aNew York :$bH.N. Abrams,$c1994.
040A $S##$a1 volume (unpaged) :$bcolor illustrations ;$c22 cm
```

Output record of `csfn_apt_remove`:

```
031T ^_S10$Franco$hfilmstrip$Ca biography$cPaul Preston.
032I $S##$aNew York$bH.N. Abrams$c1994.
040A $S##$a1 volume (unpaged)$bcolor illustrations$c22
```

13 Converting to UTF8: csfn_utf8norm

In the case where the original input data is not in the UTF8 character set and it is desired that the resulting database records are in the UTF8 character set, it is necessary to use the conversion program csfn_utf8norm.

Be aware that conversion to utf8 must occur before database storage (csfn_storenorm) and also before validation (csfn_valnorm), but not before format conversion (csfn_fcvnorm).

```
*** Usage for csfn_utf8norm:

This program reads normalized PicaPlus records from stdin, converts
them to the UTF8 character set, inserts the *01U tags, and writes the
converted records to stdout.  Input data must either be encoded in the Pica
character set or the UTF8 character set, but note that for UTF8 input
the -u option must be used to prevent unpredictable "double conversion".
Records that produce errors are written to the error file.
System- and program-level errors are written to stderr.

-u              Input data is already encoded with the UTF8 character set

The following options are common to all normalization programs:

-o <origin>     Identifies the origin of the input file.
                Used as a key to get the master record. Maximum length is 9.
-i <filename>   The filename of the original import file. Maximum length is
                256.
-e <errorfile>  Records that fail will be written to this file.
-b <badsequencefile> Sequence numbers of records that fail will be written
                to this file.
-r <reprocessfile> The sequence numbers to be reprocessed will be read
                from this file.
-s <statisticsfile> Statistics of program performance will be written
                to this file.
-a              Actual run: use log files, perform other serious actions.
-p              Print to output stream only records that process
                successfully.
```

14 Title validation: csfn_valnorm

The program csfn_valnorm is a filter for validating normalized title data. In the output of csfn_valnorm, titles that fail validation are written to the error file and are replaced by empty records in the stdout stream.

Validation includes either syntactical validation or semantical validation or both. Syntactical validation is performed with the Pica+ conversion syntax key specified by the –c option. Semantical validation is performed with the validation table key specified by the –k option. To imitate the validation of CBS online, supply both the –c and the –k option.

Note that the csfn_valnorm refuses to run if neither a conversion syntax nor a validation table is supplied.

```
*** Usage for csfn_valnorm:

This program reads a file with normalized, pica-format records from stdin,
validates the syntax of each record, and writes the correct records to
stdout.
Records that produce errors are written to the error file.
System- and program-level errors are written to stderr.

NOTE that this program does not consider the value of the Unicode Master
Switch but instead considers the character set encoded within each record.

-k <key>          Key of the VAL table to use for semantical validation.
-c <key>          Key of the conversion syntax to use for syntactical
                  validation.
-n                Records being read are entry types other than titles.

The following options are common to all normalization programs:

-o <origin>       Identifies the origin of the input file.
                  Used as a key to get the master record. Maximum length is
                  9.
-i <filename>     The filename of the original import file. Maximum length
                  is 256.
-e <errorfile>    Records that fail will be written to this file.
-b <badsequencefile> Sequence numbers of records that fail will be written
                  to this file.
-r <reprocessfile> The sequence numbers to be reprocessed will be read
                  from this file.
-s <statisticsfile> Statistics of program performance will be written
                  to this file.
-a                Actual run: use log files, perform other serious actions.
-p                Print to output stream only records that process
                  successfully.
```

An example usage:

```
$ csfn_valnorm –c FCV#pica#pica#title -k VALTAB#title -e error.txt -b sequence.txt \
      –s statistics.txt < stdinfile > stdoutfile 2> stderrfile

$ csfn_valnorm –c FCV#pica#pica#entry -k VALTAB#entry -n -e error.txt \
      -b sequence.txt –s statistics.txt < stdinfile > stdoutfile 2> stderrfile
```

15 Title storage

There are several tools available for storing records to the database, each of which has its own particular uses.

15.1 csfn_storenorm

The program csfn_storenorm is meant to be the last step in title conversion: it reads normalized title data, with correct Pica+ tags, and puts it in a Sybase database. The database is chosen according to the table entry retrieved with the master record key, set by the –o option.

For titles that cannot be properly processed, the behaviour of csfn_storenorm is slightly different that the other programs in this tool set, and depends upon whether csfn_storenorm is being run in “actual run” mode or not. If “actual run” mode is being used, erroneous titles are written to the error file and nothing is written to the target database. However, if “actual run” mode is not being used, erroneous titles are written to the error file and an empty record is written to the stdout stream.

The diagnostics flag -d prints diagnostic information to the statistics stream.

Each title has several tags checked for content:

1. If necessary, tag 208@ subfield $a is inserted with today’s date.
2. Tag 042@ subfield $c is inserted with the import filename, subfield $d with the record’s sequence number, and subfield $e with the date on which the title was processed.

The program csfn_storenorm will search for the table entry MM#<origin>#IMPORT-FORMAT for information regarding the target fileset, user identification, and source data. Note that “<origin>” is replaced by the code passed in on the –o command-line option. It is vital that this import-format table exists: csfn_storenorm cannot be operated without it when running in “actual” mode.

Also, csfn_storenorm searches for the table entry ACTABL#CAT#csfn_storenorm for information on some database policies. Definition of this table is optional, however, and csfn_storenorm can operate properly on default settings if the table is not defined.

Extra import options
Additional import-format options that only apply to csfn_storenorm:

language
The default language is "EN".

link_mnemo
The mnemo code or "\\<ikt>" that is used for retrieving ikt information. The default value is "\\1019" for origin "SU", "APPUI" and "FMESH", otherwise this option is unset.

link_specification
The specification of the subfield that contains the index search phrase. The default value is "006Z $0" for origin "SU", "APPUI" and "FMESH", otherwise this option is unset.

Note: \link_mnemo and \link_specification must both be configured in order to trigger an index phrase search that checks if the input title is present in the database.

The following options only apply to origin "SU" and are used for updating tag 044B and 047S of suppressed database titles:

contents_044BS (default value "##")
contents_044Ba (default value not set)
contents_047SS (default value "##")
contents_047Sa (default value not set)

Note: Both \contents_044Ba and \contents_047Sa must be set explicitly; if absent, tag 044B and tag 047S will not be updated.

If the origin is none of "SU", "APPUI" and "FMESH" and both \link_mnemo and \link_specification are defined, the database title is updated in case the index search returns one hit.

For all origins, the input title is reported and skipped in case an index search returns more than one hit from the database. Skipped titles are written to the error stream.

Search prefix
When an index search is performed, the prefix string specified by the –t option is prepended to the search term that is retrieved from the \link_specification subfield: “<prefix> <term>”. Notice the white space between the prefix and the term, which is added by the program.

Multi iln import
If the input title contains one are more 101@ tags with an owner iln in $a, the main level record is stored with Owner ID, and the local/copy records are stored with the iln taken from the appropriate 101@ $a.

Statistics per level
In addition to the regular statistics output, the following counters are written per level:

```
csfn_storenorm  Read: a         Errors: b         Empty: c         Output: d

Cumulative level count:
Main        found: e         read: f         stored: g         skipped: h         failed: i
Local       found: j         read: k         stored: l         skipped: m         failed: n
Copy        found: o         read: p         stored: q         skipped: r         failed: s
```

Notes

1. Read (a) - Empty (c) == found (e);
2. stored + skipped + failed == found
3. A ‘skipped’ record does not mean the record is in error, as indicated by csfn_storenorm ‘Errors’.
4. If the if the input title is corrupt, only the main level counters are updated.

```
*** Usage for csfn_storenorm:

This program reads a file with titles from stdin and writes
the titles to the database or stdout.
System- and program-level errors are written to stderr.

-u <userkey>      Use to override the default user number.
-t <prefix>       Search term prefix, e.g. 'NETF' or 'DUTCHESS'.
-w                Don't create tag 042@.

The following options are common to all normalization programs:

-o <origin>       Identifies the origin of the input file.
                  Used as a key to get the master record. Maximum length is
                  9.
-i <filename>     The filename of the original import file. Maximum length
                  is 256.
-e <errorfile>    Records that fail will be written to this file.
-b <badsequencefile> Sequence numbers of records that fail will be written
                  to this file.
-r <reprocessfile> The sequence numbers to be reprocessed will be read
                  from this file.
-s <statisticsfile> Statistics of program performance will be written
                  to this file.
-a                Actual run: use log files, perform other serious actions.
-p                Print to output stream only records that process
                  successfully.
-d                Print diagnostic data to <statisticsfile>.
```

An example usage:

```
$ csfn_storenorm -o DDB –i infile –e errorfile –b sequencefile -s statisticsfile \
     < infile
```

15.2 Bulk storing of copy records: csfn_storecopies

This program is an efficient tool for storing copy records in a title database containing only the main level records to which these copy records belong. It reads a file in the normalized format from stdin, and stores the records in this file in the correct location in the title database. No output is written to stdout; the usual statistics and error files are produced.

The program assumes the following about the input file:

- It is in normalized format,
- It only contains copy records,
- Every record has a dummy 003C tag with the supplier code of the main level record to which this record belongs in the $a subfield; this code is used to search for the matching main level record in the title database,
- Every record has a dummy 101@ tag with the ILN of the record in a $a subfield.

The program finds the correct ipn for every copy record by searching the Pica indexes of the title database for a matching supplier code. Then it updates the copy number and epn of the copy record, and appends a 203@ tag to the record with a $0 subfield containing its epn, before storing the record in the database.

Note that when running multiple instances of this program you should ensure that the epn ranges of the copy records are different, by means of the -c option. Note also that a bulk mode has been added to the program. Although the old, slow mode still exists, bulk mode (enableb by the -f option) is the preferred mode of operation.

```
csfn_storecopies: store copy records in database not containing any

*** Usage: csfn_storecopies -t [-n] [-h] [-c] [-f]

Options:
-t <idxtype>            The index type with the IDs linking main and copy records
-n <number>             The number of records read. Default: all
-h                      This help screen
-c <start_epn>          The copy records are numbered starting with this
                        epn. Default: use the master file to determine epns
-f                      Fast (bulk) mode. Default: store one record at a time
                         (slow)

The following options are common to all normalization programs:

-o <origin>      Identifies the origin of the input file.
                 Used as a key to get the master record. Maximum length is 9.
-i <filename>    The filename of the original import file. Maximum length is 256.
-e <errorfile>   Records that fail will be written to this file.
-b <badsequencefile> Sequence numbers of records that fail will be written
                 to this file.
-r <reprocessfile> The sequence numbers to be reprocessed will be read
                 from this file.
-s <statisticsfile> Statistics of program performance will be written
                 to this file.
-a               Actual run: use log files, perform other serious actions.
-p               Print to output stream only records that process successfully.
```

15.3 Bulk storing of records: csfn_storebulk

This program is a very efficient way of storing title records in the database. It reads a file in the normalized format from stdin, and stores the records in this file in the title database. No output is written to stdout; the usual statistics and error files are produced. If the titles do not contain PPNs and/or EPNs yet, the program generates them from the master file; if PPNs / EPNs are present it uses those. The program will fail if a record with the same PPN or EPN is in the database already.

The use of csfn_storebulk is recommended only when the size of the input data set is very large. Otherwise, use of csfn_storenorm is recommended. Also note that running multiple instances of csfn_storebulk that update the same database leads to undefined behaviour and should be avoided. Besides, the expected performance gain of this would be minimal.

```
csfn_storebulk: stores normalized titles in bulk mode (i.e., fast).
Usage: csfn_storebulk [-t] [-h]

Options:
-t              trace output
-h              show this help screen

The following options are common to all normalization programs:

-o <origin>      Identifies the origin of the input file.
                 Used as a key to get the master record. Maximum length is 9.
-i <filename>    The filename of the original import file. Maximum length is 256.
-e <errorfile>   Records that fail will be written to this file.
-b <badsequencefile> Sequence numbers of records that fail will be written
                 to this file.
-r <reprocessfile> The sequence numbers to be reprocessed will be read
                 from this file.
-s <statisticsfile> Statistics of program performance will be written
                 to this file.
-a               Actual run: use log files, perform other serious actions.
-z               Lean-and-mean mode: do not update logs or other tables
                 (Only useful when -a is used)
-p               Print to output stream only records that process successfully.
-x               Reserved for range filtering
-y               Reserved for range filtering
```

15.4 Merging directly into the CBS database: csfn_storemm

This application will insert records directly into the CBS database (either by merging or creating new records). In contrast to the other Title Storage tools, this avoids the need to store records to an intermediate database. As such, this tool is based on the Matching-and-Merging software and therefore has some non-standard features when compared to the other Import Tools.

The “origin” code (-o option) and the “material” code (-m option) combine to create keys for required configuration tables: these entries must exist in the Sybase standard table for the application to run.

1. MM#<origin>#<material>#MMTAB
2. MM#<origin>#IMPORT-FORMAT
3. EVAL#<origin>#<material>

There are three flags related to “actual mode” that require some explanation. First, the standard “-a” flag turns on the “actual mode,” meaning that title records will be written to the CBS database and log records will be written to the log tables. The “-z” option switches off the writing of log records. The rather unusual “-t” option turns off the writing of title records to the CBS database; this behaviour is useful if the records are to be processed further by other applications before being written to the database.

The printing and output options require some explanation as well. There is a long list of printing services available via the “-q” option which are the multiple-column output format from the old csf_mm application. This output is easier for the user to examine, but it cannot be used for further processing. In the case where the output is to be processed further by other normalized-file tools or import tools, the “-q” option should not be used: the results of the “resolve” process will then be written in normalized format to standard out. Finally, the “-g” option allows the user to specify a file into which all records that are “marked” for further review are written in normalized format.

Further information about the “-q” printing options and the configuration tables are beyond the scope of this document and will not be included here. Please refer to the matching-and-merging documentation for further information.

```
*** Usage csfn_storemm -o -m [-u][-v][-q][-i][-e][-r][-s][-a][-z][-p][-h]

This is the direct-mode matching-and-merging application.
Normalized input records are read from stdin and are inserted
or merged into the a CBS database.

Options:
-h                Display this help screen
-o <string>       Origin code
-m <string>       Material code
-u                Input data is encoded in UTF8
-t                Turn OFF database transactions
                  (relevant only when -a is used)
-g <filename>     Write 'MARK' records to this file
-q <specs>        Printing specifications: (comma separated list)
                  Note: if the -q option is not used then the output
                  will be the results of the Resolve process, written
                  in normalized format.

Printing options:
<specs>: <spec> [, <spec>]
<spec>: either <mode> or <option>
<mode>: one of the following:
                  N:  No printing
                  R:  print DBS requests
                  P:  print one resolve result per IPN
                  L:  print one resolve result per IPN-ILN combination
<option>: <type> = <maxcount>
                  format = + | <fcv format>
                  diacritics = hex | octal | decimal | copy | <char>
                  nonprintables = hex | octal | decimal | copy | <char>
                  unicode = escapes | copy | graphemes
```

```
                  width = <number>
<fcv format>: pica3 | unimarc-cataloguing | usmarc-exchange | ...
<type>: one of:
                  nohits: signals without dbase hit
                  hit   : signals without cand
                  cand  : signals without match
                  match : signals with match in 2 columns
                  merge : signals with match and merge result
                  mark  : signals that are mark-ed
                  new   : signals that are new
                  newb  : signals that are new(b)
                  drop  : signals that are drop-ed


The following options are common to all normalization programs:

-a                Actual run: use log files, perform other serious actions.
-b <badfile>      Sequence numbers of records that fail will be written
                  to this file.
-e <errorfile>    Records that fail will be written to this file.
-i <filename>     The filename of the original import file. Maximum length is 256.
-o <origin>       Identifies the origin of the input file.
                  Used as a key to get the master record. Maximum length is 9.
-p                Print to output stream only records that process successfully.
-r                Reserved.
-s <statsfile>    Statistics of program performance will be written
                  to this file.
-x                Title sequence number to start at (range filtering)
-y                Title sequence number to end at (range filtering)
-z                Lean-and-mean mode: do not update logs or other tables
                  (Only useful when -a is used)
```

16 Title reprocessing: csfn_seqnorm

Titles that fail to be properly processed and stored into the Signal database will have to be repaired and reprocessed. However, titles that are reprocessed and stored must maintain the additional data stored in the 042@ tag as described in the csfn_storenorm section. The csfn_seqnorm program has been developed especially for this purpose. As input, it takes the file of normalized titles as written by a normalization program along with a file containing a list of sequence numbers that index into the normalized title file. Typically, this program will be used in place of a normalization program in a load/process/store sequence. Titles that cannot be properly processed by csfn_seqnorm are written to the error file and an empty record is written to the stdout stream.

```
*** Usage for csfn_seqnorm:

This program reads a file of normalized titles from stdin, and together with
a file of "bad sequence nubmers" extracts the corresponding titles and writes
them to stdout. Titles that contain errors are written to the error file.

NOTE: the options for range filtering (-x and -y) may not be used by this program.

Options:
-r <reprocessfile> The sequence numbers to be reprocessed will be read
                   from this file.
-h                 This help screen

The following options are common to all normalization programs:

-a               Actual run: use log files, perform other serious actions.
-b <badfile>     Sequence numbers of records that fail will be written
                 to this file.
-e <errorfile>   Records that fail will be written to this file.
-i <filename>    The filename of the original import file. Maximum length is 256.
-o <origin>      Identifies the origin of the input file.
                 Used as a key to get the master record. Maximum length is 9.
-p               Print to output stream only records that process successfully.
-r               Reserved.
-s <statsfile>   Statistics of program performance will be written
                 to this file.
-x               Title sequence number to start at (range filtering)
-y               Title sequence number to end at (range filtering)
-z               Lean-and-mean mode: do not update logs or other tables
                 (Only useful when -a is used)
```

An example usage:

```
$ csfn_seqnorm –a -o IDSTRING –i infile –e errorfile –r sequencefile \
        -s statisticsfile < infile > outfile 2>> stderrfile
```

17 Range Filtering: csfn_rangenorm

[NOTE: this application has regained relevance with the introduction of the title type in the normalized file format. Within the selected title range, the program allows the user to skip or select certain title types.
Even for range-filtering only, which is now part of the CSNRM library, it may still be desirable to use this application, since the library changes have not yet be incorporated in all of the data import applications.]

Sometimes it may be desirable to limit the number of records in any one file. The program csfn_rangenorm is a tool that will filter the input. In the output, only those records will occur of which the titleSequencenumber falls within the required range. If a selection of title types is given, titles within the range that have a different type will be shown as empty.

Note: to prevent empty records from being written to the output stream, it is recommended to use the –p option when running csfn_rangenorm. The -p option applies only to records which have failed to process successfully. Titles with a type that falls outside the selection are printed anyway.

```
*** Usage for csfn_rangenorm:

This program reads a file of normalized records from stdin and writes
the records that fall within the specified range to stdout. If only
a "start" value is given, then all records starting from that
sequencenumber will be written.
If the -c option is used, the selected record must match the required title
type(s). if the type does not match the title content in the output will
be empty.
Options:
-x <start>       Start outputting records from this position
-y <end>         Stop outputting records at this position
-c <ttltypes>    Allows the collection of titles by means of their type.
                 ttltype: one of <standard, signal, candidate>
                 e.g. -c candidate=FALSE. Maximum length is 512
-h               Show usage

The following options are common to all normalization programs:

-o <origin>      Identifies the origin of the input file.
                 Used as a key to get the master record. Maximum length is
                 9.
-i <filename>    The filename of the original import file. Maximum length
                 is 256.
-e <errorfile>   Records that fail will be written to this file.
-b <badsequencefile> Sequence numbers of records that fail will be written
                 to this file.
-r <reprocessfile> The sequence numbers to be reprocessed will be read
                 from this file.
-s <statisticsfile> Statistics of program performance will be written
                 to this file.
-a               Actual run: use log files, perform other serious actions.
-p               Print to output stream only records that process
                 successfully.
```

Example usage:

```
$ csfn_rangenorm –p –x 25 –y 100 –c "candidate=false" < infile > outfile
```

With the introduction of the title type in the normalized format, it becomes possible to store various types of title in one normalized file. The –c option in csfn_rangenorm allows multiple types to be specified. Thus, csf_mm may produce a file containing signals and candidates. If we run

./csfn_rangenorm –p –c "signals=true; candidates=true" < infile > outfile

All records in the output of csf_mm will be selected. Infile will have the same contents as outfile.

If, however, we run

./csfn_rangenorm –p –c “signals=false; candidates=false” < infile > outfile

Or

./csfn_rangenorm –p –c “standard=true” < infile > outfile

Outfile will contain title headers (comment and directives) but no title data.

By default, if any “=false” is found in the –c option, all title types will be selected (except the one that is specified). If “=false” is not found at all in the –c option, all title types will be skipped by default.

18 Denormalization of titles: csfn_denormalize

This program is a very simple tool to provide denormalized versions of titles. When titles are converted from one format to another, the output titles are mostly produced in a specific denormalized format, Denormalization is the conversion of a normalized title into a normal, real title. In many cases, denormalization means the removal of (PICA-specific) characters, or the replacement of characters.

An example. Below is a title in MARC normalized format:

```
000 0     cam0 22        450
005 000000000000000.000
100   a00000000d2001    ||| 1gerc0103    ba
101 0 ager
102   aDE
105   ay       000yy
200 1 aBetriebswirtschaftliche SteuerlehrefFernuniversitÑt,
Gesamthochschule, in HagenhKurs 1iGrundlagen, Steuerarten,
Besteuerungsverfahren. Kurseinheit 1.
EinkommensteuereDoppelkurseinheitfAutoren: Dieter SchneelochgWilfried
Wittstock
205   a[5. Aufl.]
210   aHagencFernunivd2001
215   aXII, 126 Sd.. cm
801  3aNLbPicacgISBD
```

The same title in denormalized format is:

```
     cam0 22        450 005 000000000000000.000100   a00000000d2001    |||
1gerc0103    ba101 0 ager102   aDE105   ay       000yy200 1
aBetriebswirtschaftliche SteuerlehrefFernuniversitÑt, Gesamthochschule, in
HagenhKurs 1iGrundlagen, Steuerarten, Besteuerungsverfahren. Kurseinheit 1.
EinkommensteuereDoppelkurseinheitfAutoren: Dieter SchneelochgWilfried
Wittstock205   a[5. Aufl.]210   aHagencFernuni
vd2001215   aXII, 126 Sd.. cm801  3aNLbPicacgISBD
```

Note that items like record or field lenghts (in MARC records) are not (re)generated (the spaces in the above denormalized title are intentional!).

For denormalization, the FORMATSPEC database record is used. The following options are retrieved from the formatspec element used:

- prs-denormalize: the format used for denormalization. For example, in the case where UNIMARC normalized records are input to this program, you would use a UNIMARC denormalization format as well;
- directory: denormalization to directory form or not;
- linebreak: include linebreaks in outputting.

We refer to the CBS Formatspec documentation for more information on the formatspec.

```
*** Usage: csfn_denormalize -f [-u] [csnrm_options]

This program performs a denormalization of titles.

Options:
-f          Name of the formatspec FORMAT element to be used
            NOTE: the program uses the desired FORMAT to
            retrieve parameters for the FCVDenormalize-function
-u          Input data is encoded with the UTF8 character set
```

```
The following options are common to all normalization programs:

-a              Actual run: use log files, perform other serious
                  actions.
-b <badfile>    Sequence numbers of records that fail will be
                  written to this file.
-e <errorfile>  Records that fail will be written to this file.
-i <filename>   The filename of the original import file.
                  Maximum length is 256                    .
-o <origin>     Identifies the origin of the input file.
                Used as a key to get the master record. Maximum length is
9.
-p              Print to output stream only records that process
                  successfully.
-r              Reserved.
-s <statsfile>  Statistics of program performance will be written
                  to this file.
-x              Title sequence number to start at (range filtering)
-y              Title sequence number to end at (range filtering)
-z              Lean-and-mean mode: do not update logs or other
                  tables (Only useful when -a is used)
```

Example call:

```
csfn_denormalize -f UNX# < titles.normalized.in \
                           > titles.denormalized.out
```

where UNX# is a FORMAT XML element defined in the FORMATSPEC database table.

19 Adding journal information to articles and abstracts: csfn_enricharticle

The program csfn_enricharticle is a re-engineered version of csfn_enrichartnorm maintaining the original functionality but with less default behaviour and better debug facilities.
The program csfn_enricharticle is designed to process article and abstract records in Pica+ format in normalized form. It reads data from standard input, links each record to a journal record from the database and adds some relevant data from this journal record to the article/abstract record before writing the article/abstract record to standard output.
There is an optional debug mode to allow the user some insight in the flow of the program for each record. This can be used to analyse the programs behaviour with various configurations.

19.1 Configuration

The program csfn_enricharticle has command line options to:

- indicate the origin of the input data
- control data flow

Further configuration parameters are retrieved from an *importformat table*, a standard table with a key derived from the origin of the input data.

19.1.1 Command line options

Command line options are the first to be processed at program start up. If this step finds an error the importformat table will not be read. The command line options handled by the program are listed in the table below:

Option		**Meaning**	**Comments**
-i <filename>	M	original import file	Maximum length is 256 characters
-o <origin>	M	Identifies the origin of the input file.	Used as a key to get import-format table. Maximum length is 9
-d	O	debug mode on	Allow user to follow flow of journal finding. Default: debug mode off
-a	O	Actual run	use log files, perform other serious actions. Default: dummy run.
-b <badfile>	O	Holds Sequence numbers of failing records	Sequence numbers of records that fail will be written to this file.
-e <errorfile>	O	Holds failing records	Records that fail will be written to this file.
-r	-	Reserved	Not implemented
-s <statsfile>	O	Statistics file	Statistics of program performance will be written to this file.
-x	O	Title sequence number to start at	range filtering
-y	O	Title sequence number to end at	range filtering
-z	O	Lean-and-mean mode: do not update logs or other tables	Only useful when -a is used

The –b, -e and –s options have no default file. When any of these options is omitted, the corresponding action is not carried out by the program.

19.1.2 Importformat Table

Once the origin is known and all command line options have been processed (see Command line options) the importformat table can be read. Any import format table as a standard table with a key: **“MM#<origin>#IMPORT-FORMAT”** with “<origin>” being the value specified with the command line option “-o”.
Example:
csfn_enricharticle –i myfile –o MYORIGIN
will read importformat table “MM#MYORIGIN#IMPORT-FORMAT
The importformat table is shared among a number of programs, with each of these programs using only a subset of all the attributes listed in the table.
In the table below, the attributes used by csfn_ericharticle are listed.

Attribute	M/O	Description	Example

abstracts	M	This format refers to abstracts.	Just Y or N
Signal	M	Dbsid used to search the index when finding Journals	1.22
checkjournalyear	M	Compare Year of Publication (YoP) from input record with YoP range of journal record	Just Y or N
adi_term	O	Adi term to restrict search results	1,4,5,6,3,5,A,B,D,E,F,G,I,X
adi_ikt	O/M	ADI ikt, mandatory when adi_term is specified	9008
adi_ft	O/M		05
adi_rt	O/M		06
search_spec	M	Tag with one or two subfields to build search key from.	006N$0 007N$l $0
search_ikt	M	Ikt to use when searching journal	\7004
olc_nl	M	Add subfield 001@ $cNL when $SNL is present in tags 006N or 007N	Just Y or N
default_bc	M	String to build tag 045Q $a with, when 045Q $a is absent in journal record. An empty string indicates that no 045Q $a is added when 045Q $a is absent in the journal record	

19.2 References

The program csfn_storearticle is a re-engineered version of csfn_storeartnorm maintaining the original functionality but with less default behaviour and better debug facilities.
The program csfn_storearticle is designed to process article and abstract records. The input records are expected to be in Pica+ format in normalized form.
It reads a record from standard input, then searches the database through the family index to determine whether the record is al ready present in the database. If no match was found in the database the record is stored. If a single match is found, it depends on the configuration whether or not the matching record is updated.

19.3 The abstract exception

When handling abstracts, another step is carried out before the abstract record can be stored. Abstracts are not directly linked to the journal but are linked to an article record. This article record may already be present in the database, or else it will be derived from the abstract record being processed. Once the article record is known a link to this article record is added in subfield 039D $9 of the abstract record. If the subfield is already present, its content is replaced, furthermore the link with the journal, added in csfn_enricharticle, is removed for abstracts, as they are linked to the journal through the link with the article that is linked to the journal.

19.4 Configuration

The program csfn_storearticle has command line options to:

- indicate the origin of the input data
- control data flow

Further configuration parameters are retrieved from an *importformat table*, a standard table with a key derived from the origin of the input data.

19.4.1 Command line options

Command line options are the first to be processed at program start up. If this step finds an error the importformat table will not be read. The command line options handled by the program are listed in the table below:

Option		**Meaning**	**Comments**
-i <filename>	M	original import file	Maximum length is 256 characters
-o <origin>	M	Identifies the origin of the input file.	Used as a key to get import-format table. Maximum length is 9
-d	O	debug mode on	Allows user insight in program flow, e.g. the linked journal is listed
-a	O	Actual run	use log files, perform other serious

			actions. Default: dummy run.
-b <badfile>	O	Holds Sequence numbers of failing records	Sequence numbers of records that fail will be written to this file.
-e <errorfile>	O	Holds failing records	Records that fail will be written to this file.
-r	-	Reserved	Not implemented
-s <statsfile>	O	Statistics file	Statistics of program performance will be written to this file.
-x	O	Title sequence number to start at	range filtering
-y	O	Title sequence number to end at	range filtering
-z	O	Lean-and-mean mode: do not update logs or other tables	Only useful when -a is used

The –b, -e and –s options have no default file. When any of these options is omitted, the corresponding action is not carried out by the program.

19.4.2 Importformat Table

Once the origin is known and all command line options have been processed (see Command line options) the importformat table can be read. Any import format table as a standard table with a key: **"MM#<origin>#IMPORT-FORMAT"** with "<origin>" being the value specified with the command line option "-o".
Example:
csfn_storearticle –i myfile –o MYORIGIN
will read importformat table "MM#MYORIGIN#IMPORT-FORMAT
The importformat table is shared among a number of programs, with each of these programs using only a subset of all the attributes listed in the table.
In the table below, the attributes used by csfn_enricharticle are listed.

Attribute	M/O	Description	Example
abstracts	M	This format refers to abstracts.	Just Y or N
Signal	M	Dbsid used to search the index when finding Journals	1.22
id_spec	M	Tag holding the journal id in the input record	006n $l $0
id_ikt	M	ikt specifying the index to search with id, effective for *small abstracts* only	\7008
updateExisting	M	Update a matching record from the database with input data	Just Y or N
updateOLCfromABS	M*	Update a matching **article** record with the article derived from an input abstract record	Just Y or N
check_supplement	M	Extra test in matching	Just Y or N
check_pag	M		Just Y or N
check_npag	M		Just Y or N
Userkey	M*	existing userkey (CBS login), the specified userkey must have sufficient cataloguing authority on the 'signal' database	M1999 CBSOFFL
conversion Source	M		
overwriteable	O	Comma separated list of sources from which existing records may be overwritten by the input data, may be empty	1999,2001

* Mandatory only with abstracts=Y

20 UNIX Scripts

The normalized file tools are bound together with a UNIX shell script, which can be used for either normal processing (using the normalization program) or reprocessing (using csnf_seqnorm). The script performs a process that starts with the original input file and ends with titles stored in a Signal database.

20.1 csfn_importconvert

Please see the document “Usage for csfn_importconvert” for a complete description of the syntax and usage of this script. Note that all files produced by this script are stored in the directory specified by the –t option. If –t is not defined, then the output files will be stored in the directory named in the –i option. Also note that for this script to work as expected, all the table entries required by the other normalized file tools must be properly defined.

```
*** Usage for /export/home/pica/bin/csfn_importconvert:

    -i <>: (*) full name of input file, including path.  If this script
           is being used for reprocessing, this should be the output file
           that was created by normalization in the first processing.
    -o <>: (*) code for origination source of input
    -b <>: full name of bad sequence file, if doing reprocessing
    -c <>: (*) file of commands to perform
    -t <>: sets target directory for output files
    -a:    actual run: writes records to database, writes data to batch log
    -d:    debugging run: intermediate output files created
    -p:    print-only: use to print commands without running them
    -s:    use to output statistics to statistics files
    -w:    use wide-screen display with printed comparisons, if applicable
    -m:    prevent double-dollaring in printed comparisons, if applicable

NOTE: options with the asterisk (*) are mandatory
```

An example usage:

```
$ csfn_importconvert -i ~/inputFileName -c ~/commandsFile -o DDB \
     -t ~/myDataDirectory/Today -s -d –a
```