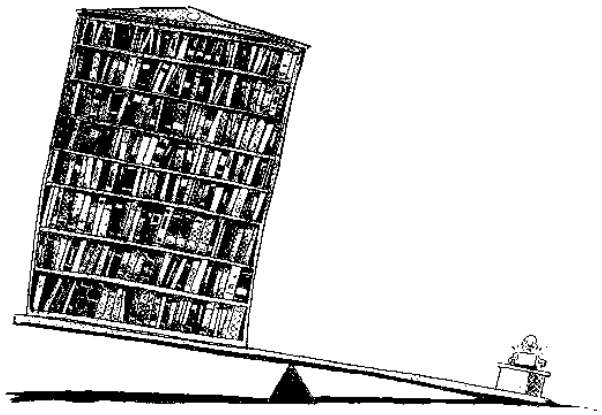
KE Texpress



SDI Guide

KE Software Pty Ltd

Selective Dissemination of Information

A new program has been developed to provide Selective Dissemination of Information (SDI) facilities. SDI enables users to register query profiles which are automatically compared with all new information entered into a database. If a match is found, various different actions can be taken.

3.1 SDI Implementation

A typical implementation of SDI is shown in Figure 2.

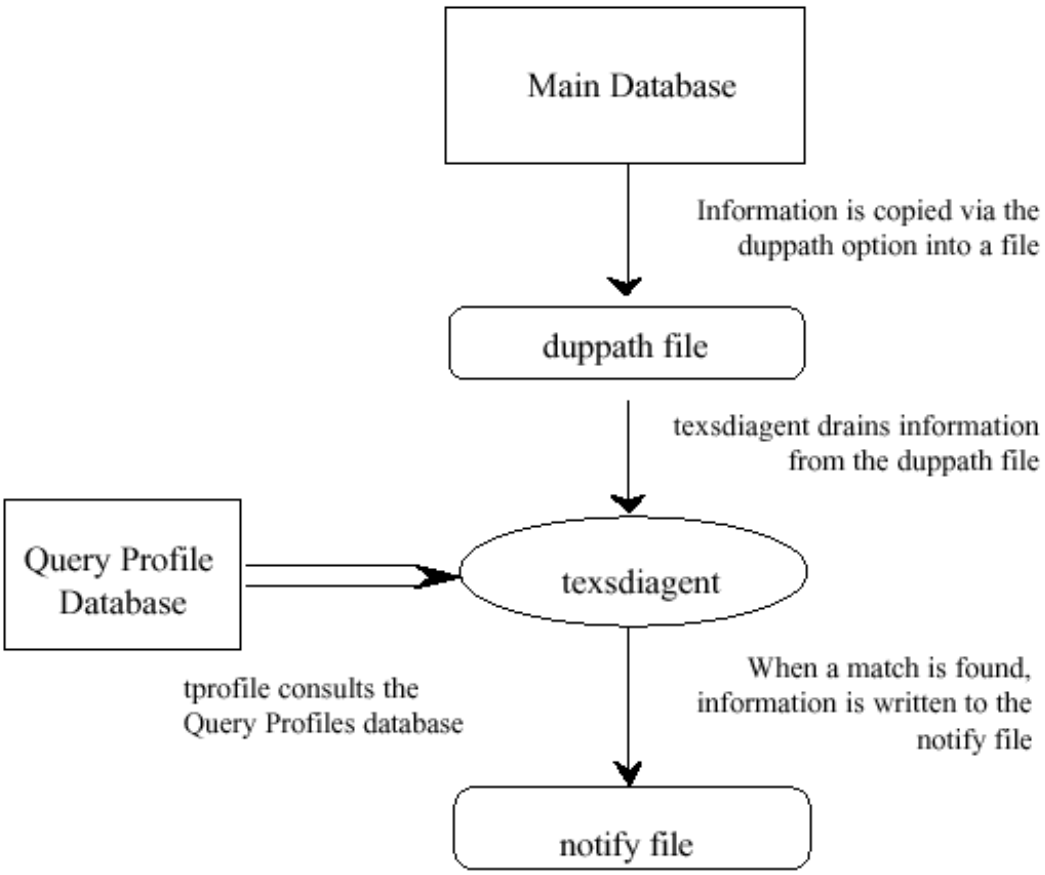


Figure 2. Selective Dissemination of Information

The basic components of this configuration are as follows:

Main Database

This is the database, typically some sort of archive, upon which SDI operates.

Query Profile Database

This is the database which stores the users' query profiles. Its Insertion form generally appears the same as the Query form of the Main Database.

Duppath File

All changes made to the Main Database are automatically copied to the duppath file using the **duppath=*file*** option which must be set on the Main Database.

Notify File

When a profile matches a record inserted into the Main Database, then information from the profile and the record is written to the notify file. Any other program can be used (or developed) to process the information loaded into this file and perform actions such as notifying users, etc.

texsdiagent

This process runs continuously (and in background) draining information from the duppath file, matching it against the stored profiles in the Query Profiles Database and, where appropriate, writing information from the Main Database and the Query Profile Database into the notify file.

The **texsdiagent** command requires several options. It can be invoked by a command of the format:

```
texsdiagent [-aform] [-pform] [-tfile] archive infile profile outfile
```

where the compulsory arguments are:

archive the name of the Main Database on which SDI is to be performed.

infile the name of the duppath file.

profile the name of the Query Profile Database.

outfile the name of the notify file.

and the optional arguments are:

-aform the name of the Report form to be used to copy information from the archive or Main Database record to the notify file. If this option is omitted, the Insertion form of the Main Database is used.

-pform the name of the Report form to be used to copy information from the Query Profile record to the notify file. If this option is omitted, the Insertion form of the Query Profile Database is used.

-tfile the name of the temporary file used while it processes the information from the duppath file. If this option is omitted, a temporary file name is derived from the name of the duppath file. This option is useful if the **texsdiagent** process should become a bottleneck (i.e. not be able to keep up with the flow

of information from the Main Database) and it should become necessary to run a second **texsdiagent** simultaneously. It is most unlikely that this option will be required.

There are several design requirements which must be satisfied before profiling can be successfully implemented. These include the following:

- The Query Profile Database must have the database option, **profile=ye**s, set. This changes the structure of the index. If this option is set on an existing database, then that database must be reconfigured and have its index rebuilt.
- The matching of profiles against records from the Main Database is performed using all items from the Query Profile Database which have the same item Id as an item in the Main Database. All other items from each database are ignored during the comparison.
- The Query Profile database must only have indexing selected for items which have the same item Id as an item in the Main Database. The indexing type **(stemmin**g, **phonetic**, etc.) and the field type attributes (such as **tex**t, **strin**g, etc.) should also be the same.
- Privilege levels of profiles and records from the main database are observed. The privilege level of the stored query profile must be less than or equal to (more privileged than) the privilege level of the record from the Main Database.

3.2 Notification

There are several methods by which users can be notified of the arrival of information matching one of their profiles. The Notify File is a text file the contents and format of which are controlled by the Report forms used as arguments to **texsdiagen**t. This file can be processed by any program. It should be noted, however, that the Notify File should be drained as it is processed and locking of this file should also be observed.

Method 1

There are several tools which assist in building a notification facility. The first is the program, **texdrai**n, which can be used to drain a file, while observing the locking on that file, and pass the information to its standard output. This program can be run on the Notify File and can be piped into another program or shell script to perform the actual notification.

The **texdrain** program can be invoked by a command of the format:

```
texdrain [-iot] [-rretries] [-sdelay] infile outfile
```

where the **infile** argument is the Notify File and the **outfile** should be – to indicate standard output. Options are as follows:

-i Lock input file before reading data.
-o Lock output file before writing data.
-t Continually check input file for data.
-sn Wait ***n*** seconds between checks on input.
-rn Try to get lock ***n*** times.

For an SDI setup the typical use of texdrain would be as follows:

```
texdrain - it infile –
```

Typically, a shell script is used to process the information passed from **texdrai**n. This script generally identifies the name of the user who owns the profile (part of the Report form from the Query Profile database used by **texsdiagen**t) and sends interactive notification and electronic mail to this user. The script can make use of any standard Unix facilities, such as **write** and **mai**l.

A tool is provided for interactive notification. This tool is similar to the Unix utility, **writ**e,. The program is called **texnotify** and instead of writing to any one of a user's terminals, it searches for the most recently accessed terminal to which it has permission to write. Thus if users are likely to be logged in on several terminals simultaneously, **texnotify** will "follow" them around as they move from terminal to terminal.

The **texnotify** command can be invoked by a command of the format:

```
texnotify [-fnrs] [-tn] user [file]
```

where the arguments are as follows:

user send the information to the Unix user, **use**r. If this user is not currently logged in, then **texnotify** silently exits.

file take information from **file** to send to the user. If this argument is omitted, the standard input is read.

-f send the information to the first terminal to which it can write, instead of searching for the most recently accessed terminal. This is likely to be a little more efficient, particularly where users are not likely to be logged in more than once simultaneously.

-n prevent the bell from ringing when the message is sent.

-r remove the file, **fil**e, after it has been sent.

-s the silent option - do not send a header identifying the source of the message. In profiling applications, the source is generally obvious from the content of the message.

-tn the number of seconds before timing out on writing to the terminal.

Method 2

Another method for notification of users is to use a third database to hold and process the notifications. This method uses **texload** with the **-t** option to continuously drain the Notify File and load records into the Notification Database. These records generally consist only of the Key value of the record from the Main Database and the Key value from the Query Profile record. These enable the Notification Database to link to and access all of the information in the appropriate query profile and the record which matched the profile.

A variety of different actions can be performed as side effects in the validation of each record inserted into the Notification Database. These actions can include copying information to a file using **copyform()** and then using the **system()** call to perform actions like **texnotif**y, **mai**l, etc.

The records loaded into the Notification Database can be retained for the user to peruse at some later date.