Recently Used Recorder user manual

Mǎ Liàng
Release 0.1
bigmaliang@163.com

April 14, 2006

Contents

1 Introduction

rur is a very simple library that offer the recently used record management interface to the POSIX [1] software developer. The record information is stored in the *~/.recently-used.xbel* file with the *Desktop Bookmarks Storage Specification* [2] compliant format. Developer can use this library to manage his recently opened file easily.

In the latest 2005, I added the recently opened file's record function to xpdf [3] with a dirty solution to store the record information into the *~/.xpdfRecently* file. After that, I recognized that this should be a common function between many software which visit one sort of files on disk(I should realize this early). So, I turned into the *gedit* [4] project's source code, and then found the *recent-file-spec* [5], which defined the common way the gnome application used to store these information. When I then read the EggRecentModel code which gedit used, I found that these code is so difficult to understand, it does a bunch of read/write operations, and it's not very optimized. On the other hand, I found ebassi is also interest on the recently used recorder function, and he has already defined the newer Desktop Bookmarks Storage Specification to store these information. So, I contacted with him and then decided to write a third-party library that *not* depend on GLib to offer this function to Non-GNOME application. This is mainly why rur present here, in addition, I want a library to record the visited file's page number for the application with an optional function, this should be very useful while you are playing with the pdf like file.

When I start to work for it, I found the hardest part is the XML file parse and write back into disk. So, I need a XML parser to do this for me. At first, I prepare to use libxml [6], but, as same as the last time, I found libxml is too difficult to understand, so I replaced it by a simpler way once again. I use *Mini-XML* [7] to parse and write the record file. Mini-XML thinks in the simplest way to parse one XML file, so I can use it in the simplest way.

To be honest, I am a very junior C programmer. rur is my first free software, with the following guy's help, it became useable eventually.

Table 1: Many thanks to

derekn	derekn@foolabs.com	rur is inspired by xpdf
James Willcox	jwillcox@cs.indiana.edu	recent-file-spec
Michael Sweet	mike@easysw.com	Mini-XML library
Stig Brautaset	stig@brautaset.org	a small and flexible linked list
Emmanuele Bassi	ebassi@gmail.com	Desktop Bookmarks Storage Specification

[1] http://en.wikipedia.org/wiki/POSIX
[2] http://devel.emmanuelebassi.net/papers/bookmark-spec.html
[3] http://www.foolabs.com/xpdf/
[4] http://www.gnome.org/projects/gedit/
[5] http://www.freedesktop.org/wiki/Standards/recent-file-spec
[6] http://xmlsoft.org
[7] http://www.easysw.com/~mike/mxml/

2 Data Structure

Currently, rur only manage some basic information of the record file. e.g. some thing like add, modify, visit time of the bookmark element defined in the DBSS(Desktop Bookmarks Storage Specification) doesn't support now. You can add it yourself or ask me to do that if necessary. Without any Graphic User Interface, rur only offer a timestamp sorted C structure list to developer as follow:

```
struct _RurItem {
        RurItem *next;

        /* attribute of bookmark element */
        char   *uri;
        int     page_num;

        /* attribute of metadata children elements */
        char   *app_name;
        char   *mime_type;
        time_t  timestamp;
        int     count;

        unsigned int status : 2;
};
```

Every member's responsibility of the structure is defined in table 2.

next	point to the next RurItem in the RurItem list
uri	file location which this RurItem represent for
page_num	the file's page number in last time application visited
app_name	application's name visited the target pointed by uri
mime_type	MIME Type of the target pointed by uri
timestamp	seconds from the system's Epoch last time visited the item
count	time counter of the application visited this file last time
status	statue of current RurItem: OLD, UPDATE, NEW, DELETE

Table 2: member's responsibility

On the other side, this structure could be better understood with a simple practical *~/.recently-used.xbel* file offered:

```
<?xml version="1.0" encoding="UTF-8"?>
<xbel version="1.0"
xmlns:bookmark="http://www.freedesktop.org/standards/desktop-bookmarks"
xmlns:mime="http://www.freedesktop.org/standards/shared-mime-info">
  <bookmark href="file:///tmp/test-213.pdf" pagenum="1">
    <info>
      <metadata owner="http://freedesktop.org">
        <mime:mime-type type="application/pdf" />
        <bookmark:applications>
          <bookmark:application name="xpdf" count="2"
           timestamp="1144827204" />
        </bookmark:applications>
      </metadata>
    </info>
  </bookmark>
  <bookmark href="file:///tmp/test-212.pdf" pagenum="1">
    <info>
      <metadata owner="http://freedesktop.org">
        <mime:mime-type type="application/pdf" />
        <bookmark:applications>
          <bookmark:application name="xpdf" count="4"
           timestamp="1144827204" />
        </bookmark:applications>
      </metadata>
    </info>
  </bookmark>
</xbel>
```

This file could be more complex, it can store many application's history recorder at once, such as gedit, Evince, openoffice.org, and so on. People who use rur must at first offer his application name and the file's mime-type [8] which he played with to rur, with these information, rur can differentiate many application's history record in one file efficiently.

3 Use rur

In section 2, we have taken a glance of the contents what rur mainly maintained. In this section, I'll describe the detail need to be pay some attention from the point of view of a rur user, during the lifetime of the your application.

[8] http://www.freedesktop.org/standards/shared-mime-info

3.1 Init

User can call function *rur_init()* to start to use rur. You offer your application's name and file's mime-type to rur, rur will return a RurItem list to you where digs from *~/.recently-used.xbel* file. All app_name member of the list equal to you offered. The list is sorted by the timestamp member, from larger to litter, this means that the most recently opened file is the first item in the RurItem list.

```
RurItem* rur_init(const char *app_name, const char *mimet);
```

In the normal situation, rur will find the record you have already made previously. But, we must also take care of the following situations:

- file *~/.recently-used.xbel* dose not exist and can't be created
- app_name or mime_type is NULL you offered

  rur will return NULL back to you.
- file *~/.recently-used.xbel* dose not exist and can be created
- file *~/.recently-used.xbel* dose not legal
- file *~/.recently-used.xbel* dose not contain any of your app_name item

  To remember your app_name and mime_type, rur will return a One-Item RurItem list to you with the item's uri is NULL, and it's app_name and mime_type is you offered.

3.2 Manage

When you have already initialed rur, you may get a RurItem list which should contain useful information for your application. For example, you may use the *uri* member of the list's item to let user select and open them again and jump to the page number last time visited. This subsection is concentrate on the management of the RurItem list. You can do the following operate on the RurItem list currently:

1. Add new item

   ```
   int rur_opened_file(const char *opend_name, RurItem **list);
   ```

   Once user opened a [new] file in your application, you can tell rur to manage the opened file's record use the function *rur_opened_file()*. The *opened_name* parameter will be stored as the uri member of the new item, the *list* parameter is the pointer point to the list pointer which is returned on subsection 3.1. The return value of this function described as following:

- −1
  opened_name is a fresh uri in *list, but add failure because of system failure. Or, parameter you input not legal.
- 0
  opened_name is a fresh uri in *list, and add to the *list as a new item in success. The app_name and mime_type is same as the other items in the *list, the timestamp is the time added the item, the count member is 1, and the status of the item is NEW(this is not so important for the user, it is used internal by rur). The value of list shall be changed in this case with the new item as it's first item(except the list is a One-Item RurItem list with the item's uri is NULL which has described in subsection 3.1, rur will reuse this item to store the new item), this is why the *list* parameter must be a pointer to a pointer.

  (By the way, The default page number is 1 of a new item. If your application is treated with some file like html that does not have page number. You can ignore this attribute of RurItem list.)
- 1
  In the *list, there is already a item with it's uri equal to opened_name you offered. So, no item will be add to the *list, but rur will update the original item's timestamp to current time, add count by 1, and change status member to UPDATE in this case.

2. Remove old item

```
int rur_delete_file(RurItem *list, const char *uri);
```

Once user closed a file or activated the remove one file's history record function you offered, you can call this function to tell rur do not manage the item no longer which the uri is equal to you offered. This function's return value also worth to be check:

- −1
  Parameter you offered to rur not legal.
- 0
  Item in list with the uri same as you offered has been deleted successful. Please note that the item hasn't been deleted actually, rur just set item's status value to DELETE for further use. So, I advise you check the item's status when you re-offer the uri list for user select to not offer the DELETE-ed item to user.
- 1
  Item delete failure, this is mostly because of the uri you offered is not exist in the RurItem list.

Currently, to remove more than one file's records from file *~/.recently-used.xbel*(for example: you want to remove all the history record you have made), you must call this function one-by-one. If necessary, you can improve this function to do such things more wiseness.

3. Change one item's page number

```
int rur_save_pagenum(RurItem *list, const char *uri, int pagenum);
```

If you call this function, it mostly like that your user have select another file to use. So, you want to save the file's page number your user used just now. Or, user want to quit from your application, you need to save current file's page number for next time you visit it.

The pagenum parameter interpreted by rur as following:

- <0
  Save the item's page_num to 1 whose uri equal to the parameter.
- $=0$
  Make the item's page_num++ whose uri member equal to the parameter.
- >0
  Save the item's page_num to pagenum actually as the parameter you specified.

This function's return value is as the same as *remove item function*, 0 represent save page number successful.

3.3 Quit

It's time to say goodbye to kernel. But before this, you need to save the list you previously initialized and managed back into *~/.recently-used.xbel*(In the mean time, don't forget to save the page number if you want). This is very easy to you, just call this function:

```
void rur_close(RurItem *list);
```

This function will do the following things for you:

- open *~/.recently-used.xbel* file for read and write
  At the end of 3.1 step, rur will close the *~/.recently-used.xbel* file, so, we must open this file again at the 3.3 step.
- lock *~/.recently-used.xbel* file
- re-load *~/.recently-used.xbel* file
  During your program's lifetime, another program may write back into the *~/.recently-used.xbel* file. Your program can't overwrite the changes that made by other programs. So, rur need read the *~/.recently-used.xbel* content again before she write you RurItem list back into this file.

- write list back into file
  rur judge the item's status member one-by-one, if it is not OLD, we'll write the item's information back into the file except the status is DELETE. rur will delete the DELETE status entry in the *~/.recently-used.xbel* file, this is the actually delete action.

  Like in section 3.1, rur treat the following situation respective:

  - file *~/.recently-used.xbel* exist but not legal
  - file *~/.recently-used.xbel* does not exist and can't be created.
    rur will *not* write the list back into disk.
  - file *~/.recently-used.xbel* does not exist and could be created
  - file *~/.recently-used.xbel* exist and have the DBSS compliant content.
    rur will write list back into disk successfully.

- unlock file
- free memory of rur

4 RUR_NOISE

Yes, rur may also produce some noises for debug use. Currently the noise is divided into 2 level: WARRING and ERROR.

Something not so painful will be print to stdout, otherwise the error message will be print to stderr. Of course, you can suppress rur to do such things by remove *-DDEBUG* in the Makefile of the rur source code.

This is all the things rur do, hope it will be useful to you, and welcome your suggestion and improvement.

5 License