

Managing WWW Browser's Bookmarks and History (a Mozilla Firefox Extension)

Ian Bugeja

Supervisor: Dr Chris Staff



Department of Computer Science and AI
University of Malta

June 2006

Submitted in partial fulfilment of the requirements for the
degree of B.Sc. I.T. (Hons.)

Declaration

Plagiarism is defined as “the unacknowledged use, as one’s work, of work of another person, whether or not such work has been published” (Regulations Governing Conduct at Examinations, 1997, Regulation 1(viii), University of Malta).

I, the undersigned, declare that the Final Year Project report submitted is my own work, except where acknowledged and referenced.

I understand that the penalties for making a false declaration may include, but are not limited to, loss of marks, cancellation of examination results, enforced suspension of studies; or expulsion from the degree programme.

Name of student: Ian Bugeja

Thesis title: Managing WWW Browser’s Bookmark and History (a Mozilla Firefox Extension)

Degree for which thesis is submitted: B.Sc. I.T. (Hons.)

Length of Thesis:	Total number of pages:	146
	Main Document:	117
	Appendix A:	4
	Appendix B:	3
	Appendix C:	21
	Appendix D:	1

Extra media: CD containing a soft copy of this report and source code of the extension.

Signature: _____ Date: _____

Abstract

All WWW Browsers offer bookmarks and history management features. The problem with these features is the low level of functionality that is offered. Many third party programs exist to offer the extra functionality, but these are not integrated with the browser and run as separate tools. The main problem with bookmark files is that they may become unorganized and grow linearly with time, possibly having duplicate bookmarks and dead bookmarks. The history is not so much used as it is too vast and complex to find a node from within it. For these reasons this thesis will present HyperBK an extension for Mozilla Firefox which will offer additional functionality to the browser's bookmarks and history utilities. Such features include automatic classification of bookmarks to their appropriate category, a thumbnail for each bookmark, verification of bookmarks and even keeping a local copy on disk in case the page is offline. On the other hand the history will be given a view where visited pages are grouped into a structure similar to the bookmark categories, where the user can apply some filters to ease his search even more. The last aspect of HyperBK will be that of integrating with search engines using them to locate a bookmarked page in case it has moved server or to provide 'See Also' recommendations for a particular bookmark category.

An evaluation was carried out to test the classification algorithm and the 'See Also' recommendations. The results obtained were quite promising as in fact 67% of bookmarks used from real users were classified into the correct category. Regarding the 'See Also' recommendations users rated them: 41% as 'Good', and another 41% as 'Really Good'.

Acknowledgements

I dedicate this thesis to my parents, Michael and Marianne, for all their love, encouragement, and support throughout my life and my education. Without them, achieving this goal would not have been possible

I would like to express my gratitude to Dr Chris Staff, my supervisor, to whom I'm indebted for his advice prior and throughout the formulation of this thesis and for the helpful comments on the text.

Table of Contents

TABLE OF CONTENTS	IV
TABLE OF FIGURES	VIII
TABLE OF TABLES	X
CHAPTER 1: INTRODUCTION	1
<u>1.1 MOTIVATION</u>	1
<u>1.2 AIMS OF THE PROJECT</u>	2
<u>1.3 THESIS OVERVIEW</u>	3
<u>1.4 ORGANIZATION OF DOCUMENT</u>	4
CHAPTER 2: BACKGROUND AND SURVEY	6
<u>2.1 INTRODUCTION</u>	6
<u>2.2 A SHORT WWW BROWSER HISTORY</u>	6
<u>2.3 LITERATURE REVIEW</u>	7
<u>2.3.1 Why and what is Bookmarked</u>	8
<u>2.3.2 Page Revisits</u>	9
<u>2.3.3 Bookmark Management Systems</u>	11
<u>2.3.4 Features required in a Bookmark Management System</u>	12
<u>2.4 PORTER STEMMING ALGORITHM</u>	14
<u>2.5 DOCUMENT CLASSIFICATION</u>	15
<u>2.6 HOW CURRENT BROWSERS PRESENT BOOKMARKS AND HISTORY</u>	17
<u>2.6.1 Microsoft Internet Explorer 7 (Beta 2)</u>	17
<u>2.6.2 Opera (v8.5)</u>	18
<u>2.6.3 Maxthon Web Browser (v1.5)</u>	20
<u>2.6.4 Mozilla Firefox (v1.5)</u>	20
<u>2.6.5 Konqueror</u>	21
<u>2.6.6 Best Browser Features</u>	21
<u>2.7 SUMMARY</u>	21
CHAPTER 3: HYPERBK SPECIFICATION AND DESIGN	23
<u>3.1 INTRODUCTION</u>	23
<u>3.2 DESIGN CONSIDERATIONS</u>	25
<u>3.3 HYPERBK EXTENSION OVERVIEW</u>	26
<u>3.3.1 Webpage Parsing and Classification</u>	26
<u>3.4 HYPERBK FEATURES</u>	29
<u>3.4.1 Bookmark Menu</u>	29

3.4.2 Add Bookmark Dialog.....	30
3.4.3 Bookmark Manager.....	30
3.4.4 Fast Bookmarks.....	31
3.4.5 The Bookmarks.....	32
3.4.6 The History.....	33
3.4.7 Storing the Bookmarks and History.....	34
3.4.8 Bookmark Caching.....	34
3.4.9 Bookmark Verification.....	35
3.4.10 Page Relocation.....	35
3.4.11 History.....	35
3.4.12 Bookmark Category See Also.....	36
3.5 THE SE REFERRER.....	37
3.6 SUMMARY.....	38
CHAPTER 4: BACKGROUND TO MOZILLA PLATFORM	39
4.1 INTRODUCTION.....	39
4.2 WHY MOZILLA FIREFOX	39
4.3 THE MOZILLA PLATFORM.....	39
4.3.1 XPCOM.....	40
4.3.2 The Chrome	41
4.3.3 Overlays.....	41
4.4 TECHNOLOGIES USED.....	42
4.4.1 XUL.....	42
4.4.2 RDF.....	43
4.4.3 JavaScript.....	46
4.4.4 SOAP and Web Services.....	47
4.5 SUMMARY.....	48
CHAPTER 5: IMPLEMENTATION	49
5.1 INTRODUCTION.....	49
5.2 INTEGRATING INTO FIREFOX.....	49
5.2.1 Building the Menus and Trees.....	51
5.3 WEBPAGE PARSING	51
5.4 WEBPAGE CLASSIFICATION	53
5.5 BOOKMARK AND HISTORY DATA SOURCES	55
5.6 ADDING A BOOKMARK.....	57
5.6.1 Bookmark Name Reduction Algorithm.....	58
5.6.2 Adding a Category	59
5.7 OPERATIONS ON BOOKMARKS.....	60
5.7.1 Bookmark Verification.....	61

5.7.2 <i>Bookmark Manager with Thumbnails</i>	61
5.7.3 <i>Bookmark Import/Export</i>	62
5.7.4 <i>First Bookmark Visit Tour</i>	63
5.7.5 <i>Divide Category Wizard</i>	63
5.7.6 <i>Bookmark Relocation</i>	65
5.8 HISTORY.....	66
5.8.1 <i>History Views</i>	67
5.9 SE REFERRER	68
5.10 SEE ALSO.....	70
5.10.1 <i>Using the SE Referrer for 'See Also'</i>	70
5.10.2 <i>Using the top Keywords for 'See Also'</i>	71
5.11 IN PAGE SUGGESTION	71
5.12 HYPERBK PREFERENCES	73
5.13 SUMMARY.....	74
CHAPTER 6: TESTING AND EVALUATION	75
6.1 INTRODUCTION.....	75
6.2 TESTING	75
6.2.1 <i>Detailed list of Tests Performed</i>	77
6.2.2 <i>Tests Summary</i>	81
6.3 EVALUATION.....	81
6.3.1 <i>Classification Algorithm Evaluation</i>	81
6.3.2 <i>Classification Results Obtained</i>	83
6.3.3 <i>See Also Evaluation</i>	85
6.3.4 <i>UI Evaluation</i>	86
6.4 CLASSIFICATION AND BOOKMARK CATEGORY SUGGESTION PROBLEMS.....	87
6.4.1 <i>Pages with no text</i>	87
6.4.2 <i>Different Categories for the same topic</i>	87
6.4.3 <i>Better Classification</i>	89
6.5 BOOKMARK RELATED PROBLEMS.....	90
6.5.1 <i>Directory Index</i>	90
6.5.2 <i>Bookmark URLs with Attribute Value pairs</i>	90
6.6 SUMMARY.....	91
CHAPTER 7: CONCLUSION AND FUTURE WORK.....	92
7.1 RESULTS ACHIEVED.....	93
7.2 AIMS ACHIEVED.....	93
7.3 FUTURE WORK.....	94
7.3.1 <i>See Also</i>	95
7.3.2 <i>Detecting Page Changes</i>	96

<i>7.3.3 Support for PDF and DOC</i>	97
<i>7.3.4 Higher UI Functionality</i>	97
<i>7.3.5 Power of the Semantic Web</i>	97
<i>7.3.6 Relation between bookmarked pages and history pages</i>	98
<i>7.3.7 Bookmark/History Sharing and Online Access</i>	99
REFERENCES	101
BIBLIOGRAPHY	105
GLOSSARY	106
APPENDIX A: FURTHER IMPLEMENTATION DETAILS	107
APPENDIX B: EVALUATION DETAILS	111
APPENDIX C: HYPERBK - USER MANUAL	114
APPENDIX D: CONTENTS OF CD-ROM	135

Table of Figures

FIGURE 1 IE7 DROPDOWN SIDEBAR WITH FAVOURITES AND HISTORY	17
FIGURE 2 OPERA ADDRESS BAR AND DROP DOWN TOOLBAR	19
FIGURE 3 HYPERBK OVERALL STRUCTURE	23
FIGURE 4 SEQUENCE OF EVENTS FROM LINK CLICK TILL WHEN WEBPAGE IS DISPLAYED	26
FIGURE 5 WEB PAGE CONTENT USED	27
FIGURE 6 KEYWORDS AND THEIR RESPECTIVE CATEGORIES.....	28
FIGURE 7 ADD BOOKMARK SEQUENCE	30
FIGURE 8 RELATION BETWEEN BOOKMARKS AND HISTORY CATEGORIES.....	36
FIGURE 9 WEBPAGES AND THEIR SE REFERRER	37
FIGURE 10 THE COMPONENTS THAT MAKE UP THE MOZILLA PLATFORM.....	41
FIGURE 11 XUL CODE AND THE EQUIVALENT STRUCTURE	42
FIGURE 12 RDF NODE GRAPH.....	43
FIGURE 13 SAMPLE RDF DOCUMENT	44
FIGURE 14 SAMPLE XUL CODE WITH TEMPLATE TAGS (TO QUERY RDF).....	45
FIGURE 15 SERVICE REQUESTER, SOAP AND WEB SERVICE.....	47
FIGURE 16 FIREFOX & HYPERBK LOAD SEQUENCE	50
FIGURE 17 LOAD EVENTS AND THEIR FIRE ORDER	50
FIGURE 18 WEBPAGE PARSING.....	52
FIGURE 19 HYPERBK WEBPAGE CLASSIFICATION ALGORITHM.....	54
FIGURE 20 BOOKMARKS DATA SOURCE NODES	56
FIGURE 21 HISTORY DATASOURCE NODES	57
FIGURE 22 ADD BOOKMARK DIALOG.....	58
FIGURE 23 CREATE NEW BOOKMARK CATEGORY DIALOG	59
FIGURE 24 BOOKMARK MANAGER.....	60
FIGURE 25 BOOKMARK VERIFICATION UTILITY	61
FIGURE 26 BOOKMARK MANAGER WITH WEBPAGE THUMBNAIL	62
FIGURE 27 LIMIT WIZARD (CREATE NEW CATEGORY)	64
FIGURE 28 LIMIT WIZARD (DIVIDE CATEGORY)	65
FIGURE 29 HISTORY VIEWER	67
FIGURE 30 SE REFERRER FINDING	70
FIGURE 31 SEE ALSO (USING SE REFERRER).....	71
FIGURE 32 SEE ALSO (USING AUTOMATIC KEYWORDS)	71
FIGURE 33 TOOLTIP OVER BOOKMARKED LINK.....	72
FIGURE 34 TOOLTIP OVER LINK WITH SAME PATH AS BOOKBAR AND LINK TITLE SHOWN UNDERNEATH.....	72

FIGURE 35 HYPERBK PREFERENCE WINDOWS.....	74
FIGURE 36 CATEGORISATION EVALUATION HITS.....	84
FIGURE 37 NUMBER OF CATEGORIES AGAINST PRECISION OBTAINED.....	85
FIGURE 38 EXAMPLE ADD BOOKMARK DIALOG WITH TABS.....	88
FIGURE 39 BOOKMARK CATEGORY TREE WITH ARROWS	89
FIGURE 40 SUGGESTED SEE ALSO METHOD (ADAPTED FROM [29])	96
FIGURE 41 HISTORY WITH BOOKMARKED PAGES & THEIR RELATIONS	98
FIGURE 42 BOOKMARK FILE SUBMISSION PAGE.....	111
FIGURE 43 SEE ALSO EVALUATION LOGIN PAGE	112
FIGURE 44 SEE ALSO EVALUATION.....	112
FIGURE 45 INSTALL CAPTION DIALOG	114
FIGURE 46 WELCOME WIZARD PAGE 1	115
FIGURE 47 WELCOME WIZARD PAGE 2	115
FIGURE 48 WELCOME WIZARD PAGE 3	116
FIGURE 49 TOUR ALL BOOKMARKS PROMPT.....	117
FIGURE 50 STOP TOUR WINDOW	117
FIGURE 51 ADD BOOKMARK DIALOG.....	118
FIGURE 52 NEW BOOKMARK FOLDER DIALOG.....	119
FIGURE 53 HYPERBK BOOKMARK MENU.....	120
FIGURE 54 HYPERBK BROWSER SIDEBAR	121
FIGURE 55 HYPERBK BROWSER TOOLBAR	122
FIGURE 56 HYPERBK BOOKMARK MANAGER WINDOW	122
FIGURE 57 HYPERBK BOOKMARK MANAGER (WITH PAGE THUMBNAILS) WINDOW.....	124
FIGURE 58 HYPERBK BOOKMARK VERIFY UTILITY WINDOW	125
FIGURE 59 DIVIDE CATEGORY WIZARD PAGE 1	126
FIGURE 60 DIVIDE CATEGORY WIZARD PAGE 2	126
FIGURE 61 DIVIDE CATEGORY WIZARD PAGE 3	127
FIGURE 62 DIVIDE CATEGORY WIZARD PAGE 4	128
FIGURE 63 HYPERBK HISTORY WINDOW	129
FIGURE 64 HYPERBK PREFERENCES PAGE 1.....	130
FIGURE 65 HYPERBK PREFERENCES PAGE 2.....	131
FIGURE 66 HYPERBK PREFERENCES PAGE 3.....	132
FIGURE 67 NO SE REFERRER PROMPT.....	133
FIGURE 68 SEE ALSO WINDOW (TOP KEYWORDS)	133
FIGURE 69 SEE ALSO WINDOW (SE REFERRERS).....	134

Table of Tables

TABLE 1 BOOKMARK ATTRIBUTES	32
TABLE 2 BOOKMARK CATEGORY ATTRIBUTES.....	33
TABLE 3 HISTORY NODES ATTRIBUTES.....	33
TABLE 4 HYPERBK USER PREFERENCES.....	73
TABLE 5 LIST OF TESTS PERFORMED	76
TABLE 6 CLASSIFICATION EVALUATION RESULTS	83
TABLE 7 'SEE ALSO' EVALUATION RESULTS	86

Chapter 1: Introduction

The use of the World Wide Web has increased tremendously in the last ten years as has the number of member web pages. Users visit this vast space through their web browsers. These web browsers need to cater for the user's browsing patterns and to help them find the correct page or document s/he requires. Two features exist in web browsers for such a use: the bookmarks and history. All browsers since the very first supported such features but unfortunately the features offered are limited and have not developed much. This thesis will produce a Mozilla Firefox Extension which will try and solve some of these problems.

1.1 Motivation

On inspection of the popular browsers i.e. Mozilla Firefox and Internet Explorer one notices that the support for bookmarks and history is quite low. In fact there are various third party programs that offer additional functionality to these browsers. The idea for this project is to integrate the two together, and have the bookmark managing utilities as part of the browser itself. This proves to be a better solution to the user as all utilities are part of one suite.

A problem which many users find is that as their bookmark file grows they find it harder to locate bookmarks inside of it (see section 2.3). This problem can be extended to the sites visited in the near past: the history list. Many sites the user visits would be related to some particular topic of interest to the user. As a topic is of interest to the user then this means that in many cases the user will have a matching bookmark category. Using this same reasoning many of the pages that the user visits can be allocated to a bookmark category. Obviously this cannot be done as the bookmark file would become enormous with lots of useless entries. But what about the history? In fact the history can be given this view, where entries would be grouped by topic instead of date/time visited or site URL.

As it is described in section 2.3.2, this representation matches more the user's mental representation, and thus previously visited URLs can be located much easier.

1.2 Aims of the project

The following are some of the aims that have to be offered in order to meet the goals and project success.

- Providing a simple way of classifying a web document into a bookmark category:

This idea is to have the appropriate bookmark category highlighted instead of adopting the approach browsers use i.e. highlighting the last category one bookmarked into (see section 3.3.1).
- Maintaining a healthy bookmark file in an organized fashion:

A bookmark file increases with time, and in many cases due to the nature of the WWW itself, bookmarked pages go offline and are lost but their link still remains in the bookmark file. Together with this a bookmark category can grow that the user might require dividing it into smaller categories (see section 5.7.1 and 5.7.5).
- Easier way of locating pages in history:

Finding a page from within the history list is quite problematic due to the nature of how the history is stored. For this reason the history will be given a new view where it will be split up by topic, where these topics will be the bookmark categories (see section 3.4.11).
- Better representation of bookmarks

The page title and URL are not always indicative of which page the bookmark represents. For this reason a thumbnail will be stored with each bookmark (see section 5.7.2).
- Portable solution

The features will be portable from one platform to another at least on the following platforms: Windows, Linux and MacOS (see section 4.2).

- Tightly coupled with browser

All features must be accessible from within the browser itself, with no external utilities so as to have everything accessible from one suite (see section 4.2).

- See Also Recommendations

These recommendations will consist of webpages similar to those found inside of a particular bookmark category. The query will be automatically computed and sent to a search engine to retrieve the results (see section 3.4.12).

1.3 Thesis Overview

This section gives a brief overview of HyperBK and its main components. HyperBK is a Mozilla Firefox Extension which can be used to replace all the bookmark and history management found inside of the Firefox browser.

The main task of HyperBK is that of automatically classifying a webpage to one bookmark category once it is loaded into the browser. This is also the new organization that is given to the visited web pages history list.

Other tasks that HyperBK performs include: suggesting the bookmark category a bookmark should fit in, maintaining a bookmark category in case there are too many bookmarks, and verification of bookmarks (i.e. checking that the web page still exists). Some features to allow the user to traverse better the bookmark list include: searching the bookmark file with keywords from the contents of the page, rather than just the page title, and a page thumbnail for each bookmark. In case that a bookmark is offline HyperBK will also load the webpage from a local cache on disk.

With respect to the new history the user will be capable of applying some filters to reduce the number of visible pages. These would include: filtering by date/time, keyword, and period visited.

The final set of features involves using search engines. The first option includes a See Also option with each bookmark category, which fetches a list of similar pages by automatically generating a query from past queries or from the automatically collected terms. The second option is that of rediscovery of a bookmarked web page in case this turns out to be offline, whereby the user can issue this command to try to relocate the bookmarked page.

1.4 Organization of Document

Chapter 2 gives a brief history of WWW browsers. Following this there is a literature review on users' browsing patterns and bookmark management systems. Tools that are used to process text, like Porter's stemming algorithm and document classification, are also described. In the last section current browsers are compared against each other on the features they offer regarding bookmarks and history.

Chapter 3 introduces HyperBK and describes the features that are offered together with its overall structure and sequence of how the events take place.

Chapter 4 deals specifically with the Mozilla platform and its main components namely XPCOM and the Chrome. Following this there is a description of the technologies used in this project: XUL, JavaScript, RDF and Web Services.

Chapter 5 deals with the implementation aspects of HyperBK. It also includes screenshots for each and every component implemented.

Chapter 6 is divided into two main parts: testing and evaluation. The testing ensured that HyperBK was functioning according to the tests performed. On the other hand the evaluation is divided into two sub parts. One deals with the web page classification algorithm while the second deals with the 'See Also' recommendations.

Chapter 7 is the concluding chapter of this thesis. It presents the results obtained and gives a list of future work that can be done to extend HyperBK to future versions.

Finally there are four appendixes which are attached to this document:

- Appendix A deals with some development details such as the URI of the RDF data sources and the list of common words that are excluded from the web pages during parsing.
- Appendix B explains how the web interface for the evaluation was and how the evaluation results were collected.
- Appendix C is the User manual to HyperBK which lists all the features and describes how to use them.
- Appendix D lists the contents of the CD-ROM.

Chapter 2: Background and Survey

2.1 Introduction

This chapter is divided into the following parts: the first section gives a brief history on WWW browsers. Following this there is a literature review on bookmarks, history and browsing patterns of users. The third section gives a brief overview of the Porter Stemming Algorithm and Document Classification. The final section describes how current browsers handle bookmarks and history.

2.2 A Short WWW Browser History

The history of browsers as we know it today can be traced back to November 1993, when Mosaic the first graphical web browser for X-Windows was released. Mosaic was developed at the National Center for Super Computer Applications (NCSA) following Tim Berners-Lee's work on hypertext and the http protocol. In fact Mosaic was tested on Tim Berners-Lee's web server. This browser being graphical introduced the support for sound, video and simple forms [1]. It also had support for user bookmarks and history of previously visited web pages. Mosaic was the tool which helped to spread the use of the WWW. A year after Mosaic's release Netscape Communication Corporation released Netscape Navigator. This browser was built on top of Mosaic and was distributed freely among students, teachers and researchers all over the world. Netscape 2 then supported frames and JavaScript while the third version introduced the mouseover features [1].

Till 1994 Microsoft had not taken the Internet very seriously but after realizing its potential they too required a browser for their operating system. Microsoft bought rights from Spyglass Inc which held the rights of the Mosaic browser, and so Microsoft released Internet Explorer [2]. IE3 was quite a decent browser as it was the first to support Cascading Style Sheets (CSS). In fact Internet Explorer dominated the

whole browser share since then as it was freely distributed with Microsoft Windows making it quite useless for a user to obtain Netscape [2].

On January 1998 Netscape announced that its browser will be distributed freely including its source code. This was the birth of the Mozilla project which was established as a non profit organization to preserve choice and innovation over the internet.

Internet Explorer dominated the browser market till around 2003 when Mozilla started slowly to eat its share. On the 9th November 2004 the Mozilla Foundation officially released the Firefox 1.0 browser. This browser is licensed under the open source Mozilla Public License and is available at no cost. At that time it offered much more functionality than its rival Internet Explorer including tabbed browsing and Rich Site Summary (RSS) integration [3].

2.3 Literature Review

The World Wide Web (WWW) has grown to vast sizes in the past 10 years. In fact this space is on the exponential increase reaching 70 million web servers in August 2005 [4]. This is one of the reasons why users browsing pages are finding it hard to be able to keep track of the pages they visit. For this reason users need a variety of tools to assist them in their daily browsing experience. In fact the two most popular features that exist to help the user to be able to keep track of the vast space of the WWW are the Bookmarks (or favourites in Microsoft Internet Explorer) (Over 92% of users have a Bookmark archive [5]) together with the browser's History, as all browsers adopt such features with only minor differences.

Bookmarks are links to web pages stored in a hierarchal tree, whereby each category of links is normally grouped into a folder. The user can use the bookmarks to revisit a page simply by clicking on the link. In this way the user will have his personalised local URL repository. The problem is that as described in [6] users don't find much help in these features. Some of these problems are due to the nature of the WWW

itself. Pages move and are constantly being deleted and updated. Although these changes happen all the time the user is not notified when such an event happens. In fact for a user to notice that a page has changed s/he has to visit the bookmark manually from time to time.

Other problems are due to the way the bookmark list is stored and structured. Each bookmark name is normally the page title, and in many cases this is not descriptive enough to remind the user what that page is about. Although the bookmark name is editable and the user can change it to his taste this does not mean that all users do so. Another problem arises when the user does not maintain and organise the bookmark file. The bookmark file grows linearly in time and over 93% of users create 0 to 5 bookmarks each browsing session [7] (This study might be misleading as it was carried out in 1997, and the user browsing skills may have changed significantly since then). Thus if not managed well the bookmark file will grow and become large in an unorganised way which will make it hard for the user to remember where each bookmark is located.

2.3.1 Why and what is Bookmarked

All users at some point in time use bookmarks as this feature is quite popular and easy to use. When a user decides to bookmark a page, then the page falls under one of these sections [5]:

- General usefulness
- Quality
- Personal interest
- Frequency of use
- Potential future use

All of them point to the fact that at some time the user wishes to revisit the page and in many cases s/he would like to be notified if the content of that page changes. Normally bookmarks lead to sites with specific content. For example, according to the study performed in [5] really few users bookmark specific news events but rather

bookmark the top root node of the news site. Similarly for search engines, few users bookmark search results, but bookmark the search engine instead.

Bookmarks are basically used for the following reasons as described in [5]:

- Reduce the cognitive and physical load of managing URL addresses
- Facilitate the return to groups of related pages
- Enable users to create a personal information space for themselves and others

Bookmarks are used in only 2.7% of all navigation actions [7] but still are quite important. They help the user remember URLs from previous browsing sessions, facilitating the return to such pages.

The problem arises when a user visits a page which s/he does not bookmark. This page can be accessed in the future by visiting the browser history. But the way the history is organised is not an easy way to find the page, especially if the user forgot the exact time when s/he was looking at it. As described in [5] users end up bookmarking pages to enable access to previous browsing sessions which leads to a larger bookmark file with no use. This would not be necessary if the history were organised and the user could view it in a more friendly way.

2.3.2 Page Revisits

Tauscher and Greenberg [8] found out that 58% of web pages a person views are pages that have already been viewed in other sessions. This means that browsers need to cater for such a high revisit percentage. Bookmarks are just one way of revisiting a page, there is the Back button (30% of all navigation methods) [8] and also the History.

The History is a list of visited URLs together with their respective page title, date and time of the last visit. The main problem with such a list is that a person needs to scan the whole list to find a desired page [9] but this is not the only problem. People have trouble recognizing the page as the title and URL are not representative enough [9].

In fact as described in [8] the Back/Forward buttons are heavily used as these go to the previously visited pages in the same session but on the other hand as history refers to all sessions it is much more complicated. This problem is shared by bookmarks too but in my opinion history suffers more due to the fact that in bookmarks the user would have specifically bookmarked a page while in history the page gets added automatically.

In [9] Kaasten and Greenberg adopt quite an interesting method. They integrate the history and bookmark together by creating implicit bookmarks by the number of visits a page has. Such pages have a vertical bar on the side which indicates the number of visits and the user can easily click on a page to automatically bookmark it. They also offer search features so that users can rapidly filter and search through the history. Such filters are by visit count, keyword in title and domain. In fact there is a lot of similarity between the bookmarks and the history due to the re-visitation patterns that occur as there is a 58% probability that the next visited page has already been seen [8]. This actually means that there is a huge similarity between the bookmarks which represent the topics the user normally visits and the history which is the total of all pages visited.

Although filters help the user to navigate through the history it is still a problem to find the required item immediately. This is due to the fact that the history is organised by time or alphabetically and this does not match the user's mental model [10]. In fact a more sensible way would be to organise it by project or topic. In [10] two methods for organizing web history were tried out. In the first method a web page importance measure was given to each page. This measure was calculated by the amount of keystrokes, mouse activity, similarity of web pages with URL and same session pages were regarded to be similar. Hierarchical clustering is then used to cluster the similar pages using their similarity ratings. In the second method a modified "Hebbian" learning rule was used to adjust the similarity of pages.

In my opinion one cannot say that pages in the same session are similar, particularly nowadays with the advance of tabbed browsing. Many users could in fact be

browsing on more than one topic at the same time. In fact the tabbed browsing features increases the complexity required to monitor the user's movements through hyperspace. A new page can be opened in the same tab, a new tab or even a new window. This also increases the number of open pages as a window can hold many tabs.

As the history is quite vast it is daunting to try a manual organization [11]. Picking a page from history at random in many cases might not be indicative of what the page was visited for [11]. This becomes truer as the entry in the history of a particular page gets older. This proves the fact that an automatic method of classifying is required for the history which would automatically place the document in the most matching category.

2.3.3 Bookmark Management Systems

A number of bookmark management systems are present nowadays although none of them are integrated with a WWW browser in any way. In fact they all exist as third party tools, which is probably why such managers are not so popular. These managers range from online websites [12] [13] which enable a user to store his bookmarks, to machine local programs [14] [15] [16] which import browser bookmarks and manage/process them.

An example of a machine local program is HiBO [14] which can automatically organise bookmarks. HiBO works by downloading the URL and parsing the html document. It then extracts the thematic words using a lexical chaining technique which is used in page summarisation. These thematic words are mapped to a hierarchy of categories. This system has a fixed hierarchy structure into which each page should fit. This is quite a disadvantage with regard to other systems as the user cannot create a personal hierarchy. After the hierarchy has been built HiBO offers the following functionality:

- Searching by topic, site/domain or keyword
- Organising Bookmarks in a hierarchal format
- Sorting the bookmarks in the hierarchy's topics

Another variety of systems are online web applications where the user has to log into such websites and is presented with his links [12] [13]. These sites are particularly useful as they enable two features which local bookmark managers cannot offer. One is that the bookmarks are available throughout the globe and the other is that bookmarks can be shared (if the user wants). Other features include automatic detection of duplicate bookmarks where the user is notified if s/he adds a bookmark to a page already bookmarked. [13] offers a tour through a bookmarked category by presenting the user with the first bookmarked site and then switching to the next/previous by clicking on a 'Next/Prev' buttons. Yet such systems do not offer features such as automatic organization of the bookmarks into categories. All the categorization has to be done manually by the user. Also these systems do not mark any dead bookmarks.

Check&Get [15] is an Internet Organiser and Web-Monitoring system. This tool organises the bookmark list and monitors each page for modifications. Highlighting the content that has changed on the page is quite useful for the user who wants to quickly see what has been updated since the previous time s/he has visited the page. In fact this is quite useful for web pages that have their content change on a daily basis. This system also offers to cache the pages locally for offline reference.

Another system that is available to store WWW links is BookMap [16]. This system instead of giving the simple plain look to the bookmark lists presents it in a graphical manner like a graph called BookMap. This gives the user a better view of his collection of bookmarks so that s/he is able to spot the one s/he wants immediately. In addition, instead of using the normal way of displaying a link (i.e. the page title) this system takes a snapshot of the page and displays this as a small icon.

2.3.4 Features required in a Bookmark Management System

After the above survey a number of important features can be pointed out about the features required in a bookmark management system. In fact Abrams, Baecker and Chignell [5] list the following necessary features:

- *Organization*

The effort that the user requires to organise the bookmarks should be the minimum possible. This can be done either by providing an automatic “filing mechanism” or else by having “automated sorting capabilities”. This ensures that the bookmark list is always organised and thus when a bookmark is required it is found easily and efficiently.

- *Visualization*

As time passes by the bookmarks increase and thus in many cases there are too many bookmarks in one category to be able to show them to the user in one screen. This is quite a problem as the user would like to spot the bookmark s/he requires immediately instead of having to scroll and search for it. The ideal solution is to have a representation whereby all the bookmarks in one category are visible to the user inside one window without the need of scrolling to view them all.

- *Representation*

In many cases the title and URL of the bookmark are not sufficient to remind the user why that page was bookmarked. In fact this is why BookMap [16] uses the title together with a dump (small screenshot) of the page to label it. The chance of the user remembering the layout of the page is greater than the page title itself. Another technique that can be used is to add to the representation of the bookmark terms that occur inside the document. This can enable searching of bookmarks by keyword search.

- *Integration*

The user will be using the browser to browse the WWW. Many bookmark management systems are external programs to the browser. In fact this in many cases might lead to duplicate or non-synchronised bookmark lists. Together with this the user is required to open a different program from his browser to be able to access the web sites s/he wants. Ideally there is closer integration between a bookmark management system and the WWW browser.

Each of the features described above have underlying tasks which are much more complex problems. First of all to be able to automatically insert the web page into the

correct category in the bookmark list then the document would have to be examined by some document classification algorithm. This would have to match the document with the correct category. There are many different methods of document categorisation. Li and Yamanishi [17] come up with document classification based on a finite mixture model. This is a powerful method but has two big disadvantages, making it unsuitable for classifying web pages into the bookmark categories. First of all it assumes that the categories are defined at start with all the keywords in the categories. Secondly there is no way to tell if a category exists for a particular document, as the algorithm would assign the document to the category with the highest ranking. This is not good in the bookmark scenario where in many cases the user would start off with a blank bookmark file (no categories) and then increase categories as the times goes by.

Another document classification method is described in [18]. This method is based on the fact that a summary of the page would be enough to classify a page, and even further by extracting the popular terms inside the page one would be able to classify the page into the required category. In this paper the technique used is an adaptation of Luhn's summarization where in this technique the popular terms of the page would be extracted (removing stop words and some web page jargon). As shown in [18] this technique produces good results.

2.4 Porter Stemming Algorithm

Porter Stemming Algorithm is used to remove automatically suffixes from a word to reduce it to its basic form. This is particularly used in Information Retrieval (IR) systems whereby lots of different words can be reduced, and thus compared with each other. This algorithm removes suffixes like 'ed', 'ing', 'ion', 'ions' etc. This means that words such as CONNECT, CONNECTED, CONNECTING, CONNECTION, CONNECTIONS will all be reduced to CONNECT. This method makes searching through an IR system much easier as if the user specifies the word connected, internally the IR system can look for all the other words which have a similar stem [19].

This algorithm was developed in 1979 in the Computer laboratory, Cambridge England by M. F. Porter as part of a large IR project and was then published in the 1980 as an algorithm for suffix stripping. The original stemming algorithm was written in BCPL a common language at that time.

2.5 Document Classification

Document Classification is a method in which similar documents are grouped together into one group. The classification can be done in two ways: either supervised or unsupervised. In the former human feedback would normally be given if a document is classified correctly or not. On the other hand in unsupervised, the classification will be done without any form of feedback being totally independent.

There are two main methods of classification: clustering and categorization. By clustering some group structure is found for a set of documents while in categorization documents are assigned to a structure known in advance.

Document Classification systems work in three phases being:

- Document representation
- Classifier construction
- Classifier evaluation

In document representation the document is represented as a series of term and the term frequency. These terms can be reduced to remove irrelevant and redundant ones a process known as dimensionality reduction [20].

The best individual features (BIF) method evaluates all words individually according to a given criterion, sorts them and selects the best subset of words. Since the vocabulary usually contains several thousand or tens of thousands of words, BIF methods are popular in TC {text classification}. However, such methods evaluate each word separately, and completely ignore the existence of other words and the manner in which the words work together. [20]

Once the document can be represented in some form this will be compared with each category or group of documents. A score will be given to each category/group. The

final step will be evaluating the scores to find the most appropriate category/group for the document.

A problem that is encountered at this point is that of efficiency. Salton [21] has emphasised this and explains a better method where a 'representative document group vector' would be selected out of each group and this will be matched with the document being classified.

One of the simplest methods of assigning a document to a cluster is to see the intersection of the terms with that cluster. This method in many cases would not consider the number of terms that are present in each group, or the non-presence of some [22].

The term frequency is the mostly used feature to determine the nature of the document. There are various techniques how documents can be classified, like:

- Naïve Bayes classifier
- Latent semantic indexing

Naïve Bayes classifier

A simple probabilistic classifier is the naïve Bayes classifier which is based on probability models. Naïve Bayes classifier can be derived from Bayes' theorem and uses the maximum likelihood method. Although this method uses a very simple design it works much better in many complex real-world situations than one would expect [23].

Latent semantic indexing (LSI)

LSI is a technique invented in the 1990 which is used to make the document easier to classify and search. This technique solves two fundamental problems: when writers use different terms to refer to the same topic and words which have multiple meanings. LSI uses a document term matrix where the rows correspond to documents and columns correspond to terms. More explanation on this method can be seen in [24].

2.6 How Current Browsers Present Bookmarks and History

This section gives a brief overview on the features that most popular browsers have with respect to bookmarks and history. The browsers examined are:

- Microsoft Internet Explorer v7
- Opera
- Maxthon
- Mozilla Firefox
- Konqueror

2.6.1 Microsoft Internet Explorer 7 (Beta 2)

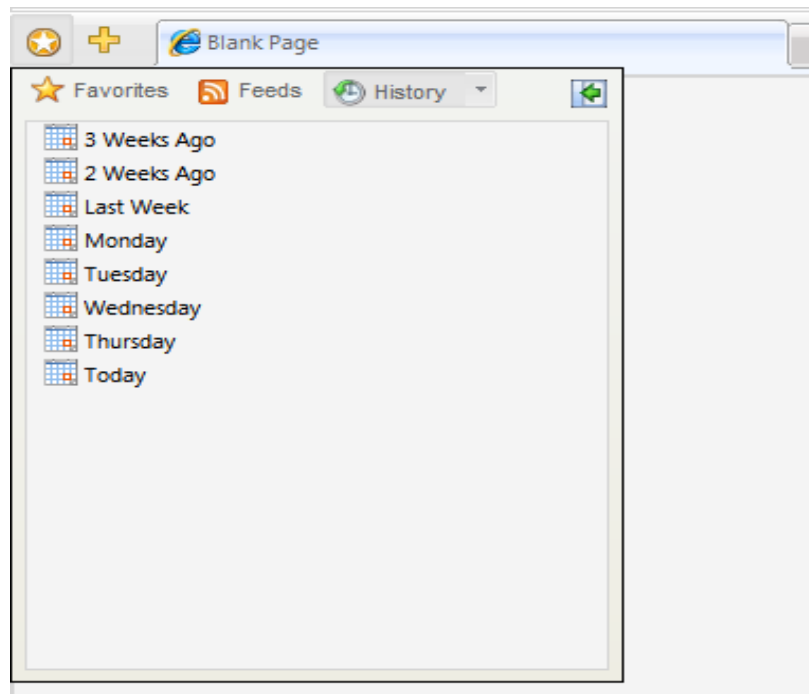


Figure 1 IE7 Dropdown sidebar with Favourites and History

This is the latest version of the most popular browser from Microsoft. It has many new features including new ways of handling/displaying favourites (bookmarks) and also supports tabbed browsing by default. As Internet Explorer has no main menu bar the Favourites are displayed in a sidebar (Figure 1) which drops down from the top. In fact this is a tree with the categories and all the favourites that are present. Beside this button there is also a button which pops out a menu to add, import/export or organize the Favourites.

The organize favourites dialog is very simple as there are only four options available through which a user can maintain organized favourites. These are:

- Create New Folder
- Rename
- Move
- Delete

The add to favourites dialog is simple too, in fact it only contains a drop down list with the categories a favourite can fit in; the favourite name and a button to create a new category. The selected category is the one where a favourite was last saved into.

The history is accessible through the same sidebar that contains the favourites. It can be viewed according to the following four parameters:

- By Date
- By Site
- By Most Visited
- By Order Visited Today

Apart from the above Microsoft Internet Explorer does not offer any more functionality regarding bookmarks and history. But one can say that it offers the bare minimum that is required to maintain an organized favourites list, although effort must come from the user to sort it and maintain it.

2.6.2 Opera (v8.5)

Opera is a descendant from the Netscape suite of browsers, and although being a really good browser it is not much used due to the fact that it was available at a cost. Opera has the Bookmarks menu in the main menu bar. This bookmark menu contains an Add Bookmark Option, and Manage Bookmarks at the top of the root menu. In other submenu there is a 'Bookmark Page' and 'Open all Folder Items' command.

The add bookmark dialog in this case has more options as one can edit the bookmark name, give a nickname to a page, modify the URL and a drop down list to select the category that the bookmark will be placed into. The category shown is the last category a bookmark was placed into. In fact in Opera has the most laborious bookmarking process.

An interesting feature (Figure 2) in Opera is that when the address bar is clicked a drop down toolbar drops down from beneath it which contains a menu with the top bookmarks, and a menu with the bookmarks just like the one in the main menu. On the other hand Opera has no sidebar which contains bookmarks or history. The history can be viewed as a separate tab. There is no way how one can filter out or sort the results. The only option that exists is a keyword search which would match with URL or page title.



Figure 2 Opera Address Bar and drop down toolbar

The bookmark manager which opens up in a new page has only add and delete options. Moving of bookmarks from one category to the other has to be done by dragging the bookmark from its parent to the new. There are a number of sort methods that can be used on the bookmark list which include:

- Sort by name
- Sort by nick
- Sort by address
- Sort by description
- Sort by created
- Sort by visited

2.6.3 Maxthon Web Browser (v1.5)

The Maxthon web browser is a variant of Internet Explorer. It has some additional features such as tabbed browsing which made it popular since Internet Explorer 6 had no such features.

The sidebar features and menu features resemble Internet Explorer 6 but in addition it has an “Add URL Here” to bookmark a page inside a particular category. The interesting feature of Maxthon is the add bookmark dialog. In fact this is a simple dialog but shows the name of the bookmark which is editable and a tree of bookmark categories instead of the standard drop down box. In my opinion this is a better approach than using a drop down list as the user will get a visual of many categories at one go.

2.6.4 Mozilla Firefox (v1.5)

Firefox retains the standard ways of handling bookmarks. In fact it has a bookmark menu on the main menu bar, and also has a bookmark sidebar which contains all the bookmarks. An interesting feature of this sidebar is that it contains a search bar at the top to locate any bookmark. The history sidebar is a different sidebar from the bookmarks one. This also contains a search bar at the top and in addition to this it contains a drop down list with the way to format the history views, which include:

- By Date and Site
- By Site
- By Date
- By Most Visited
- By Last Visited

Firefox Add Bookmark Dialog is quite simple. In fact it shows the Bookmark name which can be modified together with a drop down box of the category to fit in the bookmark. But in addition to this there is a button which when clicked will increase the dialog and show the whole tree.

2.6.5 Konqueror

Konqueror uses a much simpler approach than other browsers. In fact when the user presses the Ctrl-B key sequence the current loaded page is bookmark to the bookmark root. To add a bookmark to a particular category the user would have to navigate through the menu and find the desired category and click on Add Bookmark. This is in fact a cumbersome way of adding a bookmark as it may result in having lots of bookmarks stored in the bookmark's root.

Konqueror also has a sidebar which contains the bookmark and history in the same sidebar. It also has features to search the current view. The bookmark manager offers the basic functionality like other browsers.

2.6.6 Best Browser Features

As a conclusion one can easily point out the best features out of these browsers. First of all the bookmark menu is probably the best way of displaying the list of bookmarks and the categories that each bookmark resides in. It has become a standard feature and the menu/submenus divide the bookmarks into categories perfectly. The add bookmark dialog is fundamental too, unlike Konqueror which just adds a bookmark to the root category. In my opinion the best style for the add dialog would be to show the whole category tree instead of a drop down list. This reduces the number of clicks that a user requires to select a category and in addition by viewing the whole tree the user might be pointed to identify a more suitable category. The other two features that are useful are the sidebar and some bookmark managing tool. A sidebar with integrated bookmarks and history can make it easier for the user to take a look at the bookmarks or history while browsing. The bookmark managing tool should be simple and have all the necessary features to modify the bookmarks and move them from one category to the other.

2.7 Summary

This chapter described the problems and habits of users that are exhibited over the WWW. It also gave a description of how the browsers are designed in order to help

such interaction. In fact one can say that browsers do not help that much when it comes to bookmarks and history. As described the bookmark file becomes unmanageable with time and the history is not used by most users. This thesis will try to give solutions to some of these problems.

Chapter 3: HyperBK Specification and Design

3.1 Introduction

In this chapter the structure of HyperBK which is a Mozilla Firefox Extension is described together with the functionality that it offers. HyperBK replaces the bookmark functionality in the Firefox browser. Instead of adding the functionality to the existing bookmark features already present inside the browser, HyperBK replaces them. This is due to the fact that additional data needs to be stored with each bookmark. Although the Firefox browser is open source and this functionality can be easily entered into the browser's code this would not result in a pure extension but merely a new build of Firefox. Thus using the extension approach HyperBK can be easily deployed with a click from within a web page.

The diagram below (Figure 3) shows the basic functionality that HyperBK offers. The web page classification algorithm classifies the currently loaded page in the correct bookmark category and stores this in the history. When the user requests to bookmark a particular page the resultant category can be forecasted using this same method. Apart from this HyperBK offers better bookmark management. This includes holding a copy of the bookmarked page on disk and a thumbnail of the bookmarked page for easier recognition.

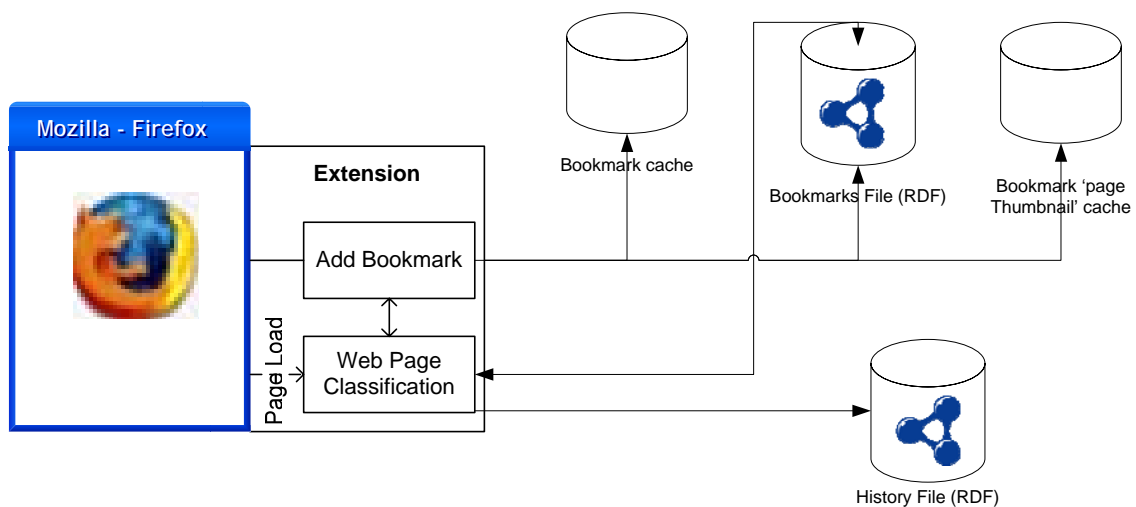


Figure 3 HyperBK Overall Structure

To be able to provide the required functionality (Figure 3), commands that are present in Firefox (such as Ctrl-D) which would add the currently loaded page as a bookmark instead need to point to HyperBK's functionality. Together with this the bookmark menu and bookmark sidebar would have to be replaced by new UI components. In addition to this some new windows are introduced which offer functionality for bookmark and history management.

The main components are:

- **Web Page Classification to Bookmark Category**
This algorithm is the core of HyperBK. It takes a web page on load and detects the matching bookmark category.
- **Bookmark menu**
This menu replaces the bookmark menu already present in Firefox. It consists of all bookmark categories and all bookmarks inside of them. Each category is divided into a submenu just like the normal bookmark menus in the browser. With functionality to add bookmark to that category, tour bookmarks in category and See Also link for category.
- **Add Bookmark Dialog**
This allows the user to add the bookmark to a particular category. The category will be selected according to the webpage classification algorithm. The user is still capable of changing this pre-selected category.
- **See Also**
Using this functionality the user is provided with a list of similar sites retrieved from a third party search engine.
- **Browser Sidebar**
This displays the bookmarks and history so the user can have a visual of them at all times. It also offers search functionality through the bookmarks/history by the name, title and page keywords.
- **Bookmark Manager**
This allows movement of bookmarks, deletion, renaming and simple browsing of the bookmarks. A thumbnail of the bookmark provides better recognition of the page that the bookmark points to.

- Bookmark Verification Utility

This sends an http HEAD request to the web server hosting the web page to check that the bookmark is still active.

- History Viewer

This viewer displays the pages inside the history as categories similar to the bookmark categories.

3.2 Design Considerations

When designing HyperBK there are a number of attributes that need to be considered which are considered to be very important and necessary for the success of HyperBK. The following are a number of objectives that have to be followed in order to achieve full user satisfaction:

Autonomy: HyperBK needs to be independent and not require user intervention for every page visit. In fact user intervention should never be requested and if an operation is required, the user should be responsible for taking the decision to do it [25].

Configurability: HyperBK should support a number of parameters which the user can modify so as to change the behaviour of HyperBK to cater for his/her needs [25].

Interaction: HyperBK should follow the normal Firefox UI layouts so as the user will not be required to learn anything new. This includes commands, icons and windows [25].

Privacy: HyperBK should keep all user information confidential, so this means that all files and folders created need to reside in the user's profile directory. When HyperBK accesses the World Wide Web no divulgence of information has to be done [25].

Resources: HyperBK should consume the least resources possible. In fact the process time to classify a page should be the minimum possible as not to annoy the

user in his/her browsing experience. Processing cannot run in the background as it has to be done while the user is browsing the web pages. In fact no processing can be done when idle as being an extension the browser will be closed once the browsing session is over.

Modification of Page Content: HyperBK should modify the page content but without disrupting its visual layout. This approach is taken as not to disrupt the page styles and colouring, thus tooltips will be preferred.

3.3 HyperBK Extension Overview

This section describes in some more detail the extension's requirements and structure, which includes the webpage parsing and classification, the interface, and finally other features offered by HyperBK.

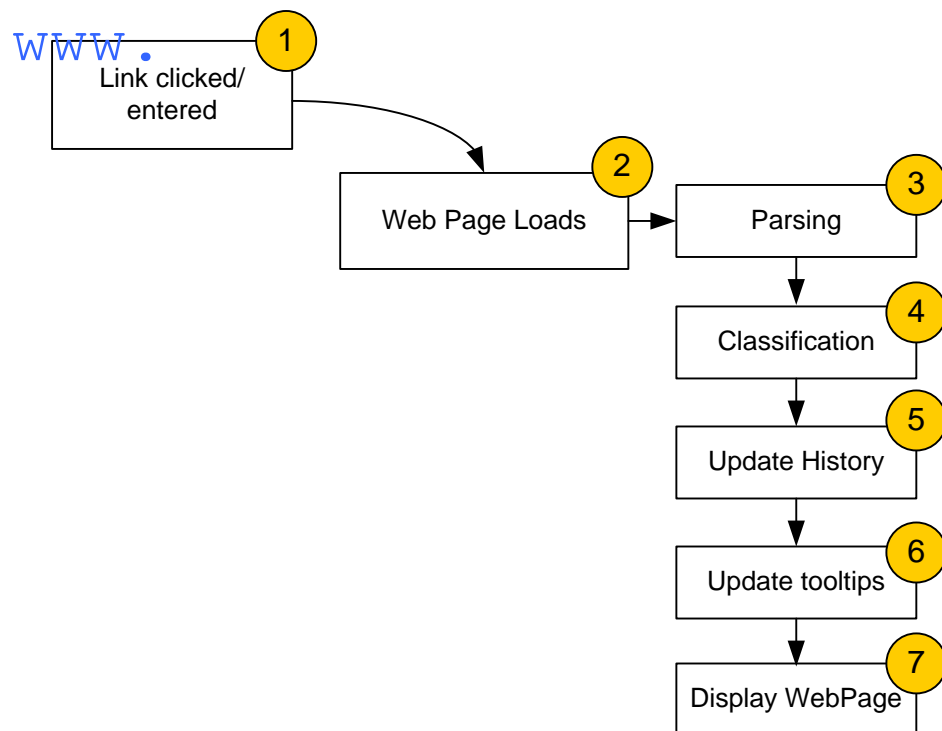


Figure 4 Sequence of Events from link click till when webpage is displayed

3.3.1 Webpage Parsing and Classification

The core of HyperBK is the webpage classification (step 3 of Figure 4). This is based on the term frequency of the textual contents of the webpage. Only webpages with

html content are considered, so other documents (like pdf, ppt, doc etc) are not classified as their structure does not form part of the browser DOM but these are loaded through third party plugins.

3.3.1.1 Webpage Parsing

The first step (step 2 in Figure 4) once a page has loaded is to parse the webpage whereby all the necessary textual content is extracted from the page. Each webpage is loaded into the browser's DOM thus this can be used instead of parsing the html string. This is a much faster way of extracting the necessary information. The extracted content consists of the following:

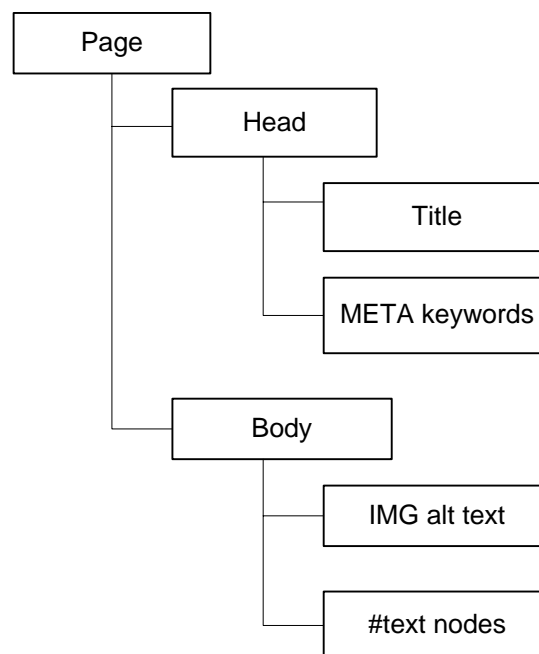


Figure 5 Web Page Content Used

This content (Figure 5) is passed on to the next step which is responsible for counting the term frequency of each keyword.

Prior to counting the term frequency the textual content has to be filtered in a way as to provide better counting, like removal of symbols and unnecessary characters. To provide more precise counting, each word is stemmed using Porter's Stemming Algorithm (ref section 2.4) so as to group similar words by comparing the word's stem instead of the whole word.

After the term frequency has been calculated then the most representative keywords can be extracted and be used to represent the document. These terms are compared with all bookmark categories to see the category the webpage is likely to fit in. (ref section 5.3 for more detail on webpage parsing)

3.3.1.2 Webpage Classification

In webpage classification the keywords that represent the webpage are compared with each category in sequence. The category is represented by a union of the keywords of each and every bookmark present in that category as shown in Figure 6. Each category is considered as an independent category consisting only of bookmarks, irrespective of the other subcategories inside of it.

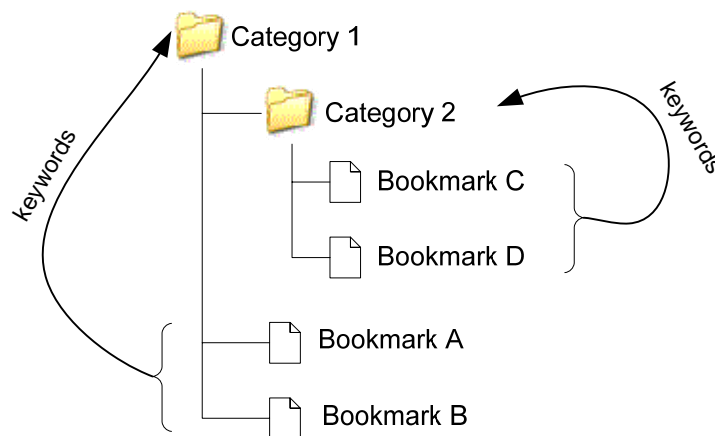


Figure 6 Keywords and their respective categories

Each category is considered as independent due to the fact that ideally there would be a clear distinction between each category. So classificationwise in the above example *Category 1* only contains *Bookmark A* and *Bookmark B*, and the union of keywords of these two bookmarks will make up the bag of keywords that represent *Category 1*. Similarly *Category 2* is made up of *Bookmark C* and *Bookmark D* and the union of keywords of these two bookmarks make up the bag of keywords for *Category 2*.

Bookmark categories that fall under the bookmark root are considered to have no association amongst themselves. Likewise this can be extended to subcategories,

although in many cases there would be an association. As an example one can have a category named 'University' and some subcategories could be: 'Java', 'Agent Technology', 'Hypertext' etc, which would represent different credits a student undertook. Although there is a relation between the categories, this relation does not relate to webpage content but to other factors.

Thus when classifying a webpage the category that has the most matches with the page keywords is the forecasted parent for that page, but this is not always the case. Sometimes the user may construct a category whereby the sites inside of it will be matching by their URL instead of their content. Although this is true one cannot decide that URL has precedence over the page contents due to the nature of the WWW and how documents are organized, where a server can host multiple websites where each site will have the same domain. A balance between the page keywords and the URL has to be done in order to distinguish such sites.

3.4 HyperBK Features

A bookmark manager needs to have high user friendly features in order to fulfil its task. One has to note that most web browser users are non-technical people who do not know anything about the nature of the WWW and how to browse and look for information. This means that HyperBK's functionality needs to be as simple as possible in order to be used by anybody.

3.4.1 Bookmark Menu

All browsers have a bookmark menu whereby each category is displayed as a submenu with the containing categories and bookmarks. If the webpage's favicon is available then this would be a good indicator with the bookmark name. The bookmark name would be the page title but the user should be capable of modifying this name to adjust it to his likes.

Within each category it would be useful to have a *Bookmark Here* link whereby the user can automatically bookmark a URL to a specific category in the fastest way

possible, bypassing the *Add Bookmark Dialog*. Another useful link is the *Tour*, which would tour a particular category. This can be found useful if the user requires finding a particular page from within a category but has no clue of which bookmark it is. Links to such features would be best if they are available at the top of each submenu.

3.4.2 Add Bookmark Dialog

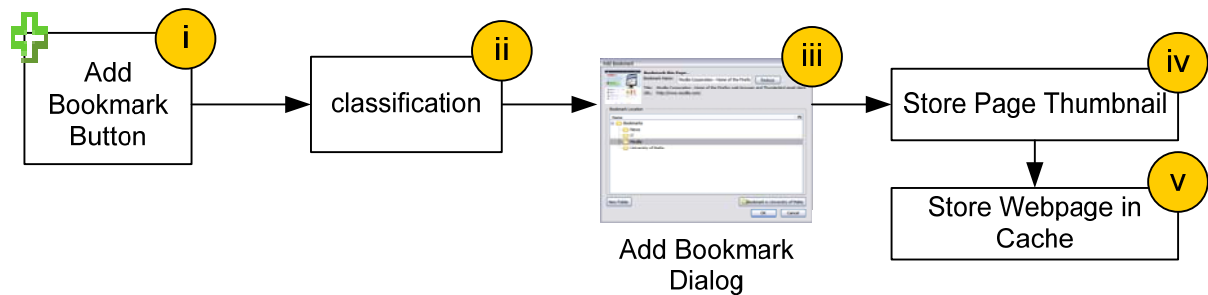


Figure 7 Add Bookmark Sequence

This dialog is shown when the user presses the CTRL-D or selects Add Bookmark command from the menu. It should allow the user to modify the bookmark name and select the destination category. The automatic classification algorithm should highlight the mostly likely category that the bookmark should be stored into. But the user should be capable of selecting a different category by clicking on any other. For this reason it would be most appropriate if all the categories are listed under each other (using a tree) instead of using the drop down box most browsers use. In many cases the user will bookmark a page which should be placed in the last category that they bookmarked into. Thus a button to provide such functionality can be useful for some users. Figure 7 shows how the add bookmark events occur, from when the add bookmark button is clicked till the actual storing to disk.

3.4.3 Bookmark Manager

A bookmark manager is used in many cases to organize the bookmarks and possibly get a deeper look on each bookmark. From the survey done in section 2.6 by inspection of the most popular WWW browsers that exist, one can conclude that a bookmark manager should offer the following functionality:

- **Create new category/folder.** With this option the user can create a new bookmark folder (category).

- **Move Bookmarks.** The user can select a bookmark and move it to a different category. As a bookmark category has fixed order inside of it then the user must be capable of moving the bookmark up/down thus changing the order of the contents of the category.
- Automatic **sorting** of the bookmarks inside a particular category or else throughout the whole bookmark file would offer great simplicity and an easy way to keep the bookmark file organized.
- A bookmark name can be **renamed**, but the URL should never be allowed to be modified. The user can easily bookmark the new URL, instead of changing the URL some particular bookmark.
- A bookmark can be **deleted** if the user decides to. No automatic deletion of bookmarks should be allowed. This is due to the nature of the WWW and how sites are sometimes offline and back online. Even if a bookmark has not been visited for a very long time is not indicative that the bookmark can be deleted as some bookmarks are stored and accessed after a long period. [5]
- To offer easier functionality the user should be allowed to apply a **filter** to the bookmark list to find some particular bookmark in an easier way. The filters should include keyword search, visit date and visit count
- A **thumbnail** of the respective page with the bookmark can offer better recognition of the bookmark. Although as explained in [28] this thumbnail does not offer proper recognition, it does help a lot.

3.4.4 Fast Bookmarks

Many times a user will start a browsing session on some particular topic or short term interest. Such visited pages do not fall as pages that are to be bookmarked but in many cases the user might still need to refer to them in the next immediate sessions. So these end up in the bookmarks list just the same, so for this reason a separate bookmark file is adopted and is called *Fast Bookmarks*. This is a flat linear list of bookmarks to which the user will add. To offer easy maintainability when the list reaches its maximum length then the first item in will be deleted so as not to have this list grow linearly. A typical operation which is useful is the “Bookmark all Open Tabs”.

In this case this function would add the open tabs to the Fast Bookmarks List instead of the standard Bookmark list.

This list can offer a solution to another situation, for instance when the user needs to close the browser but wants to save the current open web pages to resume viewing them later on. In such a scenario the user might not be sure if they need to be bookmarked or where to bookmark them. If such a feature was not present many users would bookmark them “just in case” as they might turn out to be useful and in many cases the bookmark file would end up filled with useless bookmarks or misplaced bookmarks.

3.4.5 The Bookmarks

The first step is to create a data store where bookmarks will be stored. For this reason a file which resides in the user’s profile directory is created. This file contains all the bookmarks and categories that the user creates. Each bookmark is allowed to be added to one category only. Reducing duplicates makes the bookmark archive smaller but this approach assumes that each page is on one single topic. If an already bookmarked page is rebookmarked this will result in a move of the bookmark from the other category, notifying the user of the move.

This approach was adopted to try and reduce the number of bookmarks inside of the bookmark file. It also makes it easier to classify a webpage as if duplicate bookmarks are allowed then classifying would become much more complex, as to decide where to place a similar webpage. But this approach has the disadvantage that the user is not capable of bookmarking a webpage into two categories simultaneously.

Each bookmark holds the following attributes given in Table 1:

Table 1 Bookmark Attributes

<i>Attribute</i>	<i>Description</i>
Name	Name of the bookmark
URL	URL to the page
Page Title	Title of the page that the bookmark represents

Add Date	When the bookmark was added
Last Visited	The last visit date
Visit Count	Number of visits since added
Icon	Favicon of the web site
Thumbnail image path	Path to thumbnail cache on disk
Cache path	Path to page in bookmark cache
Keywords	Keywords that represent the page

Each category in turn has the following attributes given in Table 2:

Table 2 Bookmark Category Attributes

<i>Attribute</i>	<i>Description</i>
Name	Category Name
Keywords	Union of keywords (duplicates removed) of the children in that category.

There is no limit for the number of categories a user may have but a limit to the number of bookmarks inside of a category is specified in order for HyperBK to prompt the user when the bookmark category exceeds this limit. In this case HyperBK will help the user to divide the bookmarks into subsequent categories.

3.4.6 The History

The history takes a similar approach to the bookmarks. In fact the history too is stored in a file in the user's profile directory. The categories will be identical to those found in the bookmark file. Table 3 lists the attributes each history page entry has:

Table 3 History Nodes Attributes

<i>Attribute</i>	<i>Description</i>
Name	The page title
URL	The page URL
Last Visit Date	The time and date the page was visited last.
Keywords	Page keywords
Referrer	Page URL that lead to this page
Visit Count	Number of times page was visited

Categories inside the history are much simpler as they only hold their name. All page classification is done with respect to the bookmarks and not the pages inside of the history.

3.4.7 Storing the Bookmarks and History

Bookmarks/History have to be stored on disk in order to be retrieved in the subsequent sessions. The structure of the bookmark file needs to be an efficient one so that fast searching of the bookmarks and manipulation can be performed. In fact for efficiency this bookmark list needs to be loaded in memory at startup and flushed back to disk at the end. The history needs to be very efficient too as it contains loads of entries as these add up to the total number of page visits a user makes in a 20 (approximately) day period.

RDF (ref section 4.4.2) is a versatile solution for this; first of all, all RDF documents have to be loaded into memory at startup before being modified. The structure of the file is stored as a simple ASCII text thus platform independent in the case it needs to be ported from one platform to another.

The RDF parser offers quite a great deal of functionality as one can query the data source for the required node instead of having to traverse all nodes. The structure of RDF itself makes it quite easy to maintain the bookmark/history structure where each folder can be an RDF Seq Container which contains a number of bookmarks or other containers. As the URI of each node will be the URL of the page itself, so in this manner duplicates are excluded by the underlying model itself.

3.4.8 Bookmark Caching

To avoid the situation where a bookmarked page is offline and thus cannot be accessed, a copy of the bookmarked page is stored onto disk once a page is bookmarked. This can also happen if the page moves to a different location. When the user tries to visit a bookmark page and this fails to load the copy from cache will be loaded instead, thus the user can see a copy of the bookmark at all times.

3.4.9 Bookmark Verification

From the nature of the WWW itself, web pages get taken offline all the time thus a feature to verify if a bookmarked webpage still exists would be useful. This tool should run when requested by the user and checks whether a page is online or offline and possibly checking if the document has been updated since.

3.4.10 Page Relocation

When a bookmarked page is dead or offline the user has the possibility of relocating the page by using a search engine. With this feature HyperBK will try to give a list of documents that seem to match the ones that the user is looking for. To accomplish this task the search uses the title of the offline page which should give a list of near matches.

3.4.11 History

As each web page is classified on page load then the browser can keep a list of visited URLs and their equivalent category. This is in fact the browser History which is given this bookmark view, as if all visited pages are to be bookmarked. This history is in fact a different history from the history the browser already maintains and contains all the necessary information about the visited websites.

This contains each website visited with the date and time, and the referrer of that particular page. Each page is placed in the appropriate category or else into a special category named 'Uncategorized' which contains pages that the categorization algorithm could not categorize.

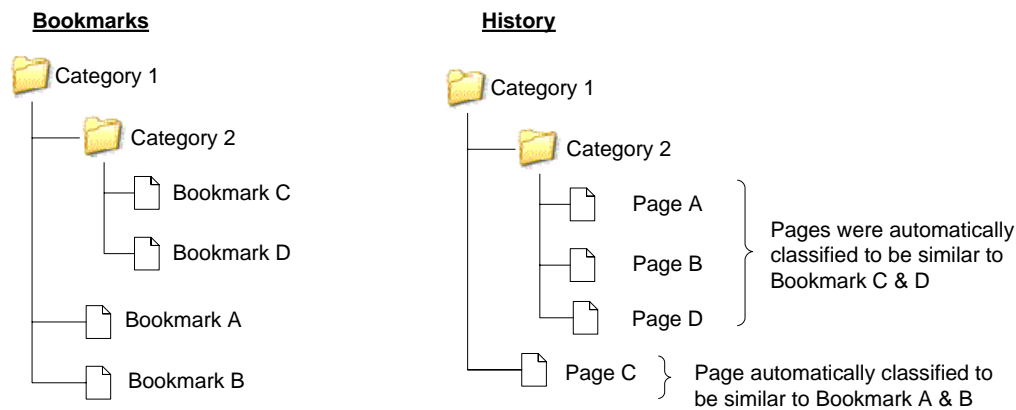


Figure 8 Relation between bookmarks and history categories

The problem with history has always been the same; it's too vast, complex and nobody remembers the exact page title and page URL that was used to refer to a particular page. In fact a user will forget most items/pages that were visited and is unlikely to recall what a random visited page was used for [11]. With this view (Figure 8) the history will match more the mental representation that the user has as it's more likely to remember that a site refers to a particular topic then remember the title or date it was visited.

Although the history is categorized by topic the user still should be allowed to apply different filters to it in order to facilitate him so that previously visited pages can be found. This filtration will include time, and date filtration but also keyword search, which includes searching through page title, URL, and even keywords.

3.4.12 Bookmark Category See Also

The user may wish to view similar websites/webpages to those inside of some particular category. To accomplish such a task HyperBK would query a search engine with an automatically generated query and get a set of results. This query generated using either of these two methods:

- The first method will use the most popular keywords from the bag of keywords that represent a particular category. Instead of performing a normal Google search this search will be performed on Google's directory which is a directory of websites divided into their appropriate category.

- The second method uses a query which is generated from past queries performed by the user. The query sent will be formed from the most popular search terms used in that category. For this reason what has been termed as SE Referrer (Search Engine Referrer) is stored with each bookmark.

3.5 The SE Referrer

A Referrer to a webpage is the previous page that was visited before the webpage. Obviously URLs entered into the address bar directly or clicking on a bookmark do not have any referrer, but those visited by following some link on a page have. Firefox maintains the correct referrer even if the link is opened in a new tab or window.

The SE Referrer (Search Engine Referrer) is the most recent search engine results page that was visited (Figure 9). Many users start browsing by visiting their favourite search engine and performing a search query. Thus when a user bookmarks a page this SE referrer is looked up and stored with the bookmark (if it exists).

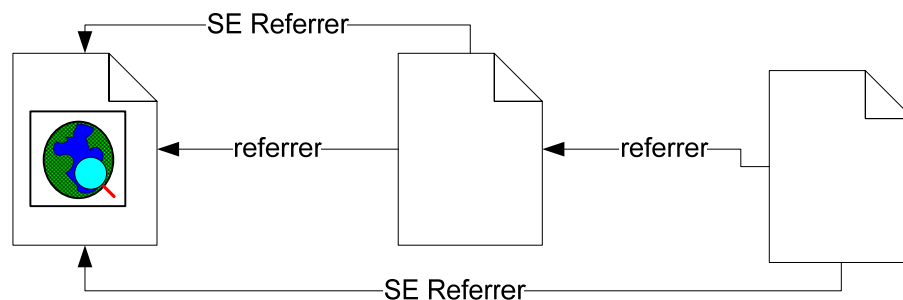


Figure 9 Webpages and their SE Referrer

This SE Referrer has other uses too; in fact many times a user might want to recall the search query used to find a particular page. Thus storing the search query is useful not only for looking up similar sites but in helping the user figure out what was the query used to find that particular bookmarked page.

3.6 Summary

In this chapter the design and specification of HyperBK was put forward. First an overview of the whole system was illustrated. Following this a more detailed description of HyperBK was given; which included the web page parsing, the classification algorithm, the bookmark/history files, and the bookmark/history management tools. Finally the SE Referrer is described and how it can be used to determine what the user is looking for.

Chapter 4: Background to Mozilla Platform

4.1 Introduction

This chapter gives an overview on the platform and technologies used to build the artefact. In the first section an overall view of the Mozilla platform and its mayor components is given. Then in the section that follows the mayor technologies used in this extension are explained namely: XUL, RDF, JavaScript and SOAP.

4.2 Why Mozilla Firefox

Mozilla Firefox is an increasingly popular web browser, currently (March 2006) holding 7% of the browser share against Microsoft Internet Explorer which has 88% [27]. These statistics basically excludes development on other browsers as their usage is quite low. The biggest advantage gained when developing for Firefox is due to the support it has for extensibility. The platform itself was made to support extensibility thus adding functionality is not complex. There is also a great amount of work going in this area and lots of extensions are being developed and distributed. Together with this one must not forget that Firefox runs on more than one platform including Windows, MacOS and Linux.

4.3 The Mozilla Platform

The Mozilla Platform provides the framework upon which Mozilla products like Firefox, Thunderbird, Camino, and the Mozilla browser itself are built. The Mozilla project started after Netscape released for free the source code of Netscape Navigator. The Netscape products prior to Mozilla did not use this platform and their structure was quite complex as in fact they did not use a layered approach. This complexity in fact limited the amount of functionality that could be offered. As a result the Mozilla platform was developed in a layered fashion in order to solve this

problem. Basically the four important features of the platform are XUL, JavaScript, RDF and XPCOM.

- XUL (XML User Interface Language) uses syntax to define the user interface.
- JavaScript is a scripting language with syntax similar to C to define functionality.
- RDF uses XML syntax to store data.
- XPCOM (Cross Platform Component Object Model) is an object discovery and management system.

All of these features make it possible to build interactive applications.

4.3.1 XPCOM

The bottom layer of the Mozilla platform is the XPCOM, which is a series of objects which can run on multiple platforms. Access to any of these objects is only available through C and C++ languages. For this reason there is another layer called XPConnect which makes it possible to use these objects from the JavaScript.

There are various XPCOM components but the most popular are those which process the XML/HTML content and provide the W3C DOM view to the XML/HTML document. Other XPCOM components are responsible with streams, and communications. The portion of the objects that are responsible with layout and rendering objects to the screen is known as the Gecko.

Each component has its own name which is made up of a contract ID followed by the version number like the following: @mozilla.org/browser/httpindex-service;1. XPCOM is very similar to COM as each component has its own type library (.xpt file) which is what makes these components portable.

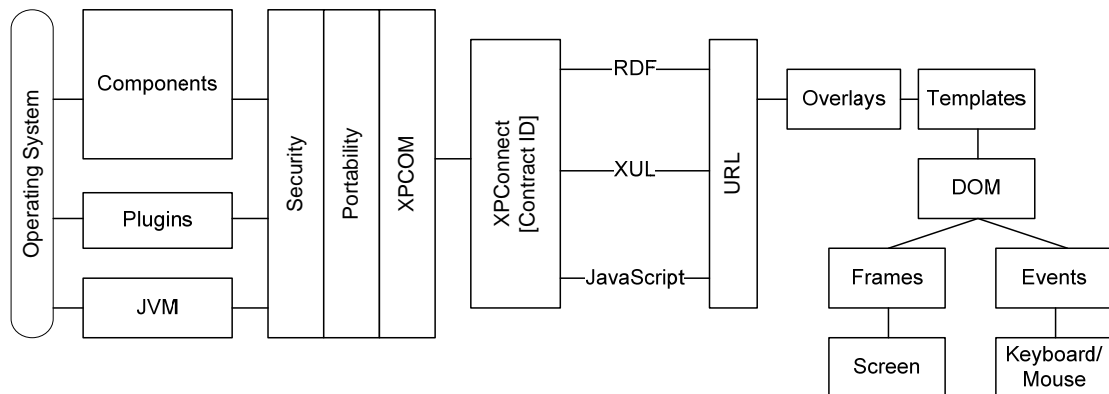


Figure 10 The Components that make up the Mozilla Platform

Figure 10 illustrates the structure of the Mozilla platform. As one immediately notices there is a split within the platform and linking the two sides is XUL, RDF, and JavaScript.

4.3.2 The Chrome

The installation of any Mozilla application is basically divided into three parts:

- The user configuration files like user/application settings, bookmarks, history etc. which would normally reside in the user profile directory.
- The system executables and libraries.
- Other application files which reside inside a location called the chrome. Such files would include data files, documents, scripts, images and other similar content. The chrome is referenced with a special URL scheme `chrome://` (just like `file://` and `http://`)

The chrome is in fact portable and in many cases the files present in it are identical for all platforms, be it a Mac, Microsoft Windows or Linux PC. This is due to the fact that inside the chrome XUL and JavaScript documents are usually stored.

4.3.3 Overlays

The power of this platform is in the extensibility and flexibility it offers. New XPCOM objects can be added and new XUL documents can be inserted into the chrome. Not only new XUL documents can be added but existing XUL documents that were written by other developers can be modified with the use of overlays.

An overlay is quite a simple feature but yet offers a lot of power in itself. It is a piece of XUL code which will be merged into another XUL document. All this merging will take place as the XUL document is loaded before its content is displayed on screen. Thus a programmer can insert his own UI elements and the JavaScript in any part of the application increasing the feature and functionality of that application.

4.4 Technologies Used

4.4.1 XUL

Mozilla applications use XUL (XML User Interface Language) to describe all the visual elements that are present on the screen. XUL uses XML syntax to define the elements. There are tags for textboxes, menus, menu items, labels, buttons and all other possible UI elements. XUL layout is guided with the use of the box tags which can have vertical or horizontal alignment.

The XUL will be loaded into the DOM (Document Object Model) which is a large in-memory structure. This DOM is not interested with the display; it is just a way of storing the XML structure into memory.

```
<vbox>
<hbox>
    <label value="Label1" />
    <label value="Label2" />
</hbox>
<hbox>
    <label value="Label3" />
    <vbox>
        <label value="Label4" />
        <label value="Label5" />
    </vbox>
</hbox>
</vbox>
```

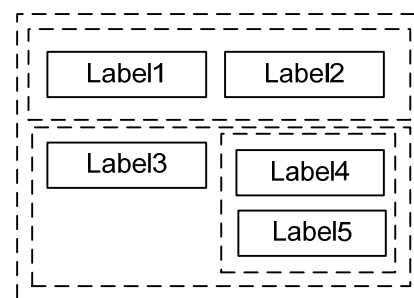


Figure 11 XUL code and the equivalent structure

The above example (Figure 11) is XUL syntax consisting of four box elements, and five labels. Elements inside the hbox have horizontal alignment thus they are

displayed one beside the other, while elements inside of a vbox have vertical alignment thus they are displayed one on top of the other.

4.4.2 RDF

RDF (Resource Description Framework) was developed to describe any object on the web. This formal data model from W3C which uses XML tags to describe its contents can be used in lots of scenarios other than to describe web resources. In fact the Mozilla platform uses RDF extensively not only for storing data but also as manifest files, jar content files and so on.

RDF provides a model for describing resources where each resource has a number of properties (attributes). Each RDF resource can be thought of as a unique object uniquely identifiable by a URI (Uniform Resource Identifier). The properties associated with each object have their respective property types and values. Each property will represent a relationship between the property value and the object itself. This value can in fact be another object or else an atomic value (string, integer etc)

RDF is based on facts which describe the data not like other data structures which have only attributes. These facts can be described using what is known as the 3-tuple, each consisting of a predicate, subject and object. E.g. (predicate, subject, object) - (isa, Search Engine, Google)

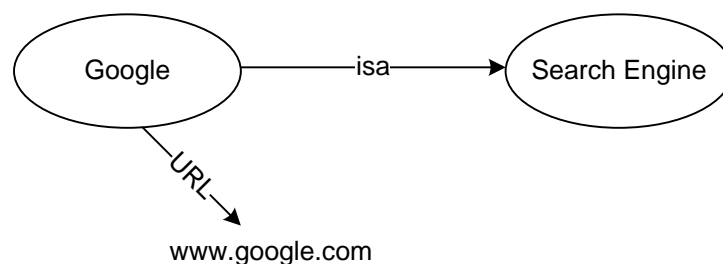


Figure 12 RDF Node Graph

An object with multiple facts can be represented using an RDF container. Thus if one looks at the above example the Search Engine can be a container with multiple instance of search engines inside of it, being Google, Yahoo, MSN and so on.

4.4.2.1 RDF Syntax

As described earlier RDF uses XML like syntax, but unlike XML there are only a limited number of RDF tags, which are:

- `<RDF>` tag which is the top root node.
- `<Description>` tag which will contain the facts to describe some object.
- Three container tags `<Seq>` `<Bag>` `<Alt>` which would contain `` tags to distinguish the items inside.

Each of the above tags can have a number of identifiers which are used to identify whole facts. They can also be used to replace literals about some object, thus instead of using a tag inside another all the literals go specified as attributes.

The three containers in RDF, the Seq, Bag and Alt hold a list of resources. The Seq represents an ordered list in which the contained items are ordered. Bag on the other hand is just a collection like a normal bag with no restrictions on the items contained inside of it. Alt stands for alternative where the contained items inside of the container are considered to be equivalent to each other. Each item inside of the container is enclosed in the `` tag.

```
<?xml version="1.0"?>
<RDF:RDF
  xmlns:NC="http://home.netscape.com/NC-rdf#"
  xmlns:RDF="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

  <RDF:Seq about="NC:DownloadsRoot">
    <RDF:li resource="C:\tmp\test_save.html"/>
  </RDF:Seq>

  <RDF:Description about="C:\tmp\test_save.html"
    NC:Name="test_save.html"
    NC:ProgressMode="none"
    NC:StatusText="Finished"
    NC:Transferred="1KB of 1KB">
    <NC:URL resource="http://www.mozilla.org/" />
    <NC:File resource="C:\tmp\test_save.html" />
    <NC:DownloadState NC:parseType="Integer">1</NC:DownloadState>
    <NC:ProgressPercent NC:parseType="Integer">100</NC:ProgressPercent>
  </RDF:Description>

</RDF:RDF>
```

Figure 13 Sample RDF document

The above (Figure 13) is a sample RDF document. It consists of a Seq Container node which contains one item. This item which has its URI (C:\tmp\test_save.html) has a number of facts such as the name, download state, mode and so on.

4.4.2.2 Querying the RDF

For RDF to be useful one must be capable of querying the data source and get the required nodes out of it. There are two ways to query the RDF and fetch its contents,

- From the XUL using the <template> tags
- From the scriptable JavaScript by using RDFDatasource XPCOM component.

Building XUL components from an RDF file is quite flexible as this enables the programmer to create UI content which is not fixed, but dynamic and which can change on the nature of the data that is stored in the RDF data source. All this is possible with the use of the template system. This query system searches through the facts in the RDF data source and returns the objects that match the search specification. This specification is enclosed in the rule tags. Thus every template consists of one or more rules where each rule will contain XUL tags for the UI elements.

```
<vbox datasources="rdf:bookmarks" ref="NC:BookmarksRoot" flex="1">
  <template>
    <vbox uri="rdf:*">
      <hbox>
        <label value="rdf:http://home.netscape.com/NC-rdf#Name"/>
        <label value="rdf:http://home.netscape.com/NC-rdf#URL"/>
      </hbox>
    </vbox>
  </template>
</vbox>
```

Figure 14 Sample XUL code with template tags (to query RDF)

The above example (Figure 14) will create a vertical box and inside of it will list of the bookmarks, first the bookmark name and beside it the bookmark URL.

In the case that the content is to be different according to the node one can specify rule tags. These rule tags can be used to match the type of a particular RDF node or

else can contain special attributes like `iscontainer` to match a container node or `isempty` to match a container node which is empty. The rule tags would be the children of the template tag and inside the rule tags there would be the XUL UI tags.

4.4.3 JavaScript

JavaScript is a member of the C family of programming languages as it shares much of the syntax and structure. In fact it is a 3GL language and contains constructs such as `for`, `while`, `if`, `switch` etc. Inside the Mozilla platform JavaScript is nicknamed SpiderMonkey and its interpreter is implemented as a C library. Although JavaScript is the well known name its official name is ECMAScript and in pre-releases it was named LiveScript [26].

JavaScript code is highly portable as it does not require to be compiled for any target machine; in fact JavaScript code is interpreted by a virtual machine like the Java VM. Unlike Java it does not require high resource requirements thus it can be used with ease in all environments. Being interpreted the variables are late-bound and weakly typed. In fact all variables are declared without any type information.

JavaScript code can be found within the script tags in XUL or else in a separate .js file being referenced in XUL through the `src` attribute in the `script` tag. This is the preferred way of having the JavaScript code, as it divides the functionality from the UI elements.

JavaScript statements are just like in C, the only difference is that a statement can be terminated by the newline character. Thus the following two statements are valid and equivalent:

```
x = x + 5
x = x + 5;
```

JavaScript supports the following native data types:

Undefined Null Boolean Number String Object

The `typeof` operator can be used to identify the type a variable holds it will return any of the above as a string. Each variable must be declared with the `var` keyword and can be declared anywhere throughout of the code.

JavaScript also supports exceptions and has the standard `try...catch` block. The `throw` keyword is used to throw an exception. This is quite useful for error handling as if the JavaScript interpreter encounters a runtime error it will return from the function it was executing.

4.4.4 SOAP and Web Services

Simple Object Access Protocol has gained much popularity over the last years. It is a method whereby objects can be exchanged between two machines over a network using XML like syntax. All the communication is done using simple ASCII text over http, which makes such a protocol much powerful as it uses an existing transport protocol which is extensively used (Figure 15).

Search engines such as Google offer a Web Services API¹. A SOAP request with the search query will be sent to Google's Web Service which in turn will return a SOAP reply with the search results to that particular query. This message can be examined processed and displayed to the user.

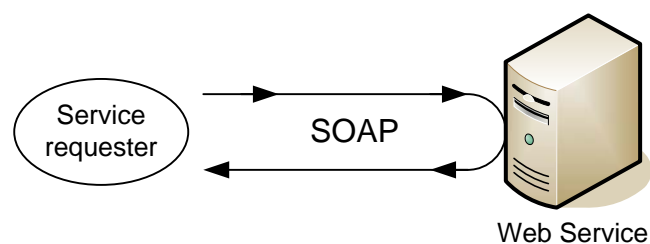


Figure 15 Service Requester, SOAP and Web Service

¹ Google Web Service API URL: <http://api.google.com/search/beta2>

4.5 Summary

This chapter give a brief overview of the Mozilla Platform and technologies such as XPCOM, XUL, JavaScript, RDF and SOAP. These technologies are extensively used in the Mozilla Platform. XUL is used for define the UI elements on screen, JavaScript the logic behind each component. XPCOM is a series of objects which form the basis of the platform. RDF is used to store data while SOAP is used to transport data from one machine to another.

Chapter 5: Implementation

5.1 Introduction

This chapter describes the implementation aspects of the HyperBK extension for Mozilla Firefox. The first section details specifically with how the functionality was integrated into the Firefox browser. Following this the classification algorithm is explained. In the last section implementation aspects of the bookmark/history management are described.

5.2 Integrating into Firefox

The integration of the bookmark and history manager functionality with Firefox is divided into two parts:

- The first part deals with hiding and disabling the old bookmark functionality and inserting the new UI elements into the Firefox window.
- The second part is to provide the page classification, bookmark and history management as pages are loaded.

The first part has to be carried out before the main Firefox window loads and displays on screen. In fact this procedure has to be performed for every new window that is opened. An overlay is used for this purpose and it contains the following UI elements:

- 'New' Bookmark Menu
- Hidden 'Fast Bookmark' Menu (which can be then enabled)
- HyperBK Toolbar
- Browser Sidebar
- Some shortcuts in the Tools Menu

When the script loads it registers an event for the window `onload` event which fires whenever a new Firefox window is opened. This event is responsible to hide the old bookmark menu and to change the `oncommand` value of a number of components,

like 'Add Bookmark', 'Bookmark All Tabs', 'Bookmark Link' and other similar functions. This event also registers an event listener for the `DOMContentLoaded` event which fires whenever a new html frame is loaded. Finally it registers an observer which fires once a whole page has completely loaded (see Figure 16).

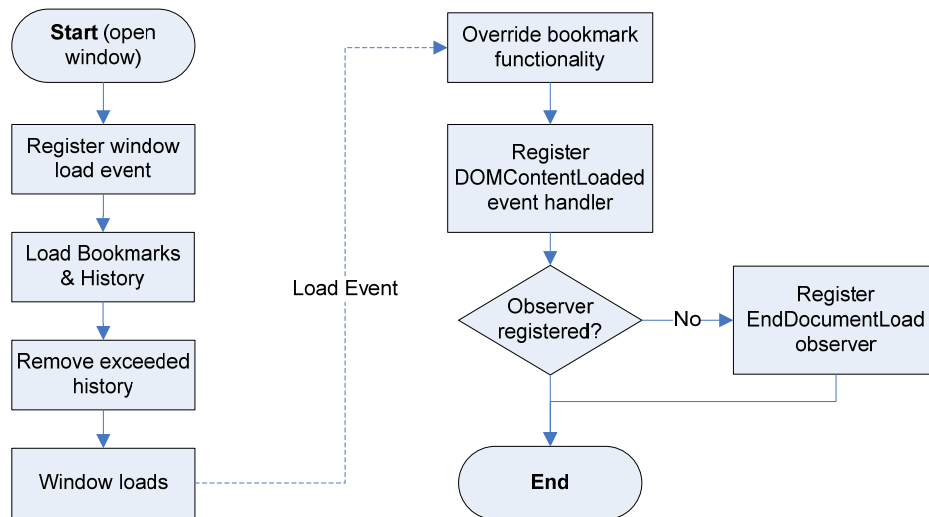


Figure 16 Firefox & HyperBK load Sequence

The way events are fired when a new html page is loaded is shown in Figure 17. First an event fires for each frame that loads, which includes any inner frames and the enclosing (outer) frame. Then the observer fires once the page has been fully loaded.

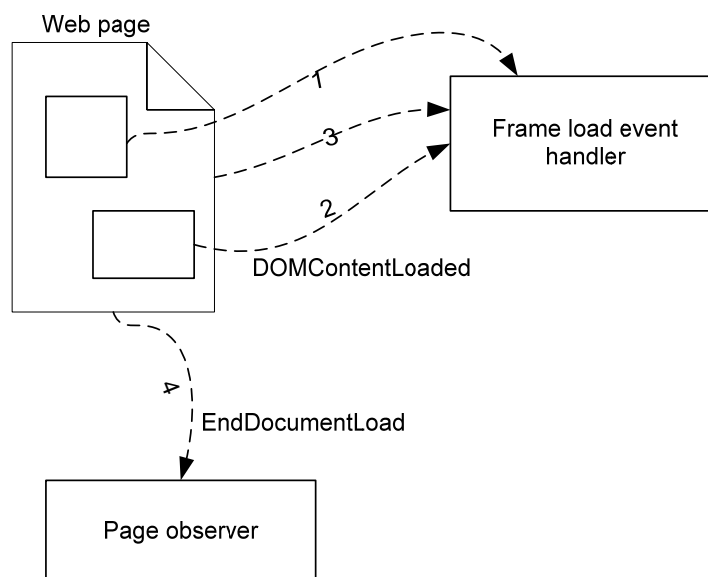


Figure 17 Load Events and their Fire Order

So whenever a frame is loaded the registered frame event handler is called. This event handler is responsible to parse the textual contents of the html page that forms that frame, and to store the keywords that represent the frame in the window DOM. A similar event is responsible to update the frame contents such as the tooltip (title attribute) of anchor tags which point to already bookmarked links. On the other hand the page observer is responsible to update the history and the bookmark file in case a bookmark is visited. This is why an observer is used as it's not directly related with updating the window itself.

The main difference between the frame load event handler and the page observer is that the user will see the page as loading while the frame load event handler is being executed but on the other hand the page observer will run in the background. Also the page observer is an XPCOM component thus common for all windows, so only one observer is registered, not one for every window like the frame `onload` event handler.

5.2.1 Building the Menus and Trees

Menus and Trees are built automatically from the RDF data source using the `<template>` tags in XUL (ref section 4.4.2.2). In fact this provides flexibility to the programmer as when changes are made to the data source the effects are reflected to all UI elements instantly. This means that multiple windows are updated by just modifying one data source common to all windows.

5.3 Webpage Parsing

The first step once a page has been loaded into the browser is to parse its contents and extract a set of keywords which can be used to represent the page and its contents so that these can be used to match the page in the ideal bookmark category. With the aid of the Document Object Model (DOM) this procedure was much simpler as the html document would already be loaded into memory into a series of objects (nodes) which form part of the DOM. The textual content required is

the text between tags, the content of some special attributes like the alt inside the img tag, but excludes text in the script tags as this would be JavaScript code.

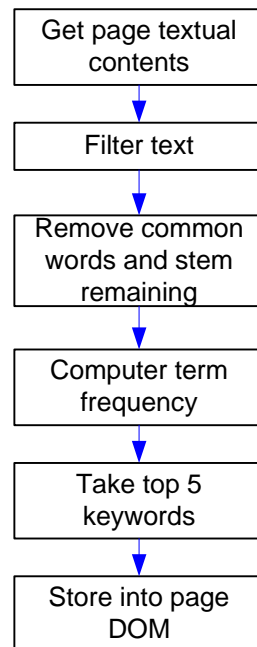


Figure 18 Webpage Parsing

Thus once a frame has been loaded, its textual content is extracted using a recursive function which will traverse the DOM recursively. The resultant string of textual content is filtered out to remove non-textual characters and to convert all characters to lowercase. Once this has been carried out the top five keywords can be extracted out of this string. This involves converting the string into an array of words and further filtering these words from any stop words or to be more exact popular English words. Removing stop words would not be enough as the algorithm works on keyword matches and there are a lot of other words commonly used which are of no relevance to the webpage's topic (see Appendix A for the list of removed words). The words that are left are stemmed using Porter's stemming algorithm and then using this stem all identical stems are grouped and counted (see Figure 18). The final result would be a list of stemmed words together with the term frequency of each word.

From this list of words only those with a term frequency (TF) greater than one are considered. The top five keywords which have the highest TF are extracted. It is expected that a page has more than one keyword with a TF greater than one. But

obviously once sorted those with the higher TF are more representative than those with less. Selecting five keywords preserves this choice but on the other hand does not result in having too many keywords for one page. In the case that the page has five or less words with a TF greater than one then only half this number is selected, as those with low TF would not be representative on the page's content.

The above algorithm is good but there is a feature present in most html documents which can produce better keywords. Many html documents have a list of META keywords in the head section of the page. These are quite indicative on the topic of the page and so they are used to boost up the keywords that are extracted from the textual content. Thus the META keywords that appear frequently inside the page contents will be selected. In this way the top five will be more indicative of the actual page topic as META keywords would have been entered by the page author.

5.4 Webpage Classification

The classification of a webpage into a bookmark category is done using the following two features of the webpage:

- The URL (more specifically the domain)
- The top five webpage keywords

Each bookmark category is represented by the union of all keywords (with duplicates removed) that the containing bookmarks have. A small enhancement that was adopted was to select only keywords that were present in more than one bookmark in the case that there was a sufficient number of keywords to represent a particular category. Thus the keywords of the webpage are compared with this bag of keywords of each category. A score is given to each category according to the number of matches, the more matches a page has, the more a page is said to match a category.

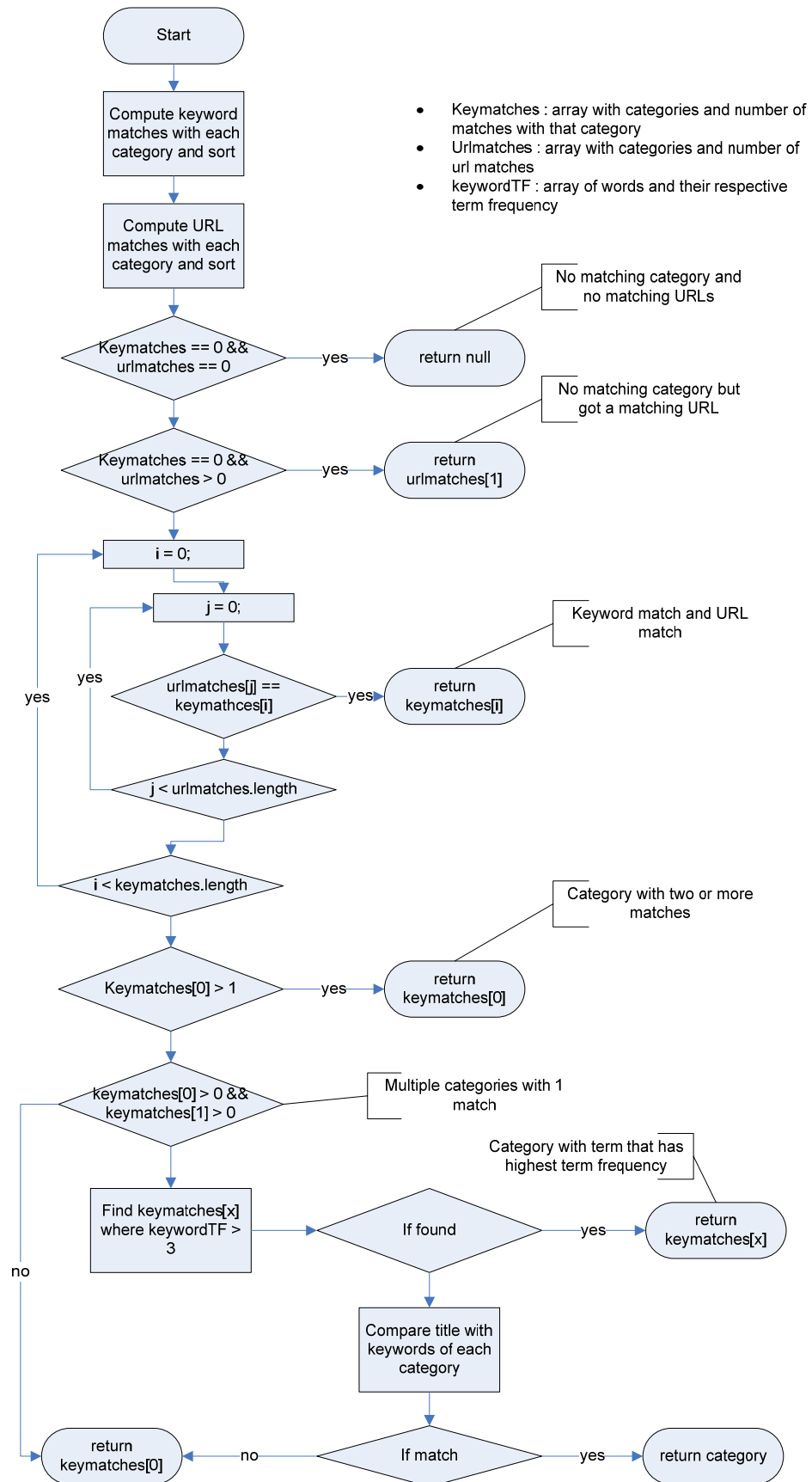


Figure 19 HyperBK Webpage Classification Algorithm

But this is not the only way how a webpage is allocated to a category; the URL plays an important role too. If a category has a webpage with the same domain as the webpage being compared with than this is considered too. The category with the highest word hit is the allocated category but if a category has lower ranking and has a URL match then this takes over. The URL match is only considered if at least one keyword match exists for that category. This is due to the fact that on the WWW different websites can be hosted under the same domain name.

If none of the above methods give any result the page title is compared with the keywords of each category. This method is used as the last way to detect a page into a category, as in fact a page with little content will always have a title and this can give a good indication of the category the webpage fits in.

(Refer to Figure 19 for the flowchart of the classification algorithm.)

5.5 Bookmark and History Data sources

The Mozilla platform has a powerful built-in XPCOM RDF service which maintains and manipulates RDF files with ease. This proved to be a powerful method of storing the bookmarks on disk. RDF is a flexible way of updating the data as the underlying service handles how the file is represented. It gives to the programmer an abstract view whereby each bookmark is represented as an RDF node, and the category which groups a number of bookmarks would be an RDF container node.

Each bookmark is represented inside of the RDF by its URL as this is unique and each folder by its name. This introduced the fact that each folder had to have a different name. Although this might seem to be a disadvantage it makes it easier for the user not to mix up categories which have the same name but reside in different folders. Each node would contain a list of literals describing the bookmark, such as the last visit date, add date, name, page title, location of cache on disk, etc. (see appendix)

Container nodes are of the SEQ (Sequence) type as order inside of the container is important (see section 4.4.2.1). No bookmark was allowed to have more than one parent as this would lead to duplicate bookmarks in the bookmark file. The idea is to keep the bookmark file as small and organized as possible. The root node is NC:BTopRoot which has only one child NC:BookmarksRoot which is the container for all bookmarks/categories that are stored (see Figure 20).

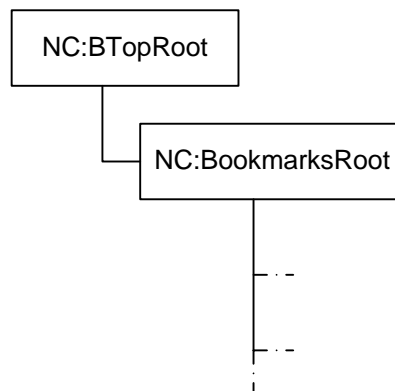
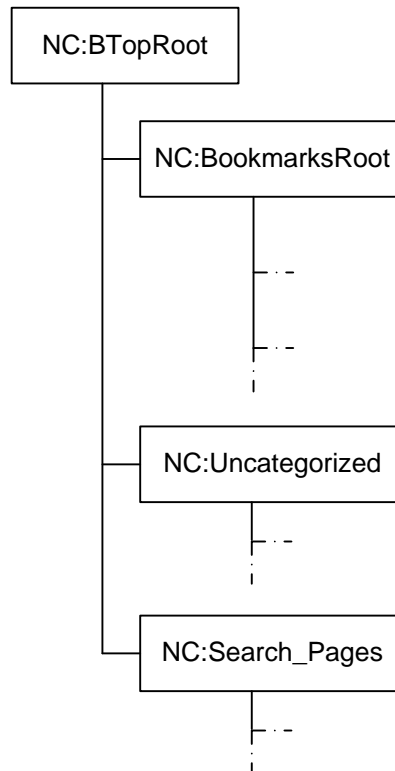


Figure 20 Bookmarks Data source Nodes

The History is quite similar in structure to the Bookmarks data source. In fact even its hierarchal structure is identical to the bookmark data source. This actually means that whenever the user creates a new category in the bookmark data source then another similar category is created in the history data source. Similar for deletion and renaming where the equivalent category in history would be updated too. The only two differences is that in the history the container nodes are of BAG type (see section 4.4.2.1) as order in the history is not important and in addition to the containers inside of the bookmark data source there are two additional containers called NC:Uncategorized and NC:Search_Pages which is the children of NC:BTopRoot (see Figure 21). In NC:Uncategorized uncategorized webpages would be placed while in NC:Search_Pages would contain search pages results. Each webpage in the history data source would have stored with it the keywords that represented that page together with the visit date and time. This visit date and time is updated once a web page is revisited.



5.6 Adding a Bookmark

The process of adding a bookmark is made as simple as possible to speed up the bookmark process. Basically there are two methods:

- One way of bookmarking a page is to click on the Add Bookmark command or pressing ctrl-D. This operation will open the add bookmark dialog which can be seen below (Figure 22). This consists of the bookmark name which can be modified; bookmark URL which is fixed, a thumbnail of the webpage and a list of folders (categories) where the bookmark can be placed into. The webpage classification algorithm will highlight the highest matching category automatically. The user can easily change this selection by clicking on another category. In the case that the webpage classification algorithm does not find a matching category then the bookmark category which was last bookmarked in will be selected. (As is done by default by Firefox, IE 7 etc (see section 2.6)). Whenever the suggestion differs from the last category bookmarked into a button to bookmark in the last category is made visible. In Figure 22 the button

“Bookmark in University of Malta” would enable the user to bookmark in that category, as it was the last category bookmarked into and as this differs from the one suggested (i.e. Mozilla).

- The second method adds the bookmark to a particular category from the Bookmark Here command at the top of each bookmark category (from the main menu). This will automatically bookmark the page in that category without opening any dialogs. It's a fast way of bookmarking a page without the need to modify anything. This method does not use the classification algorithm as the user will click the respective Bookmark Here link in the destination category.

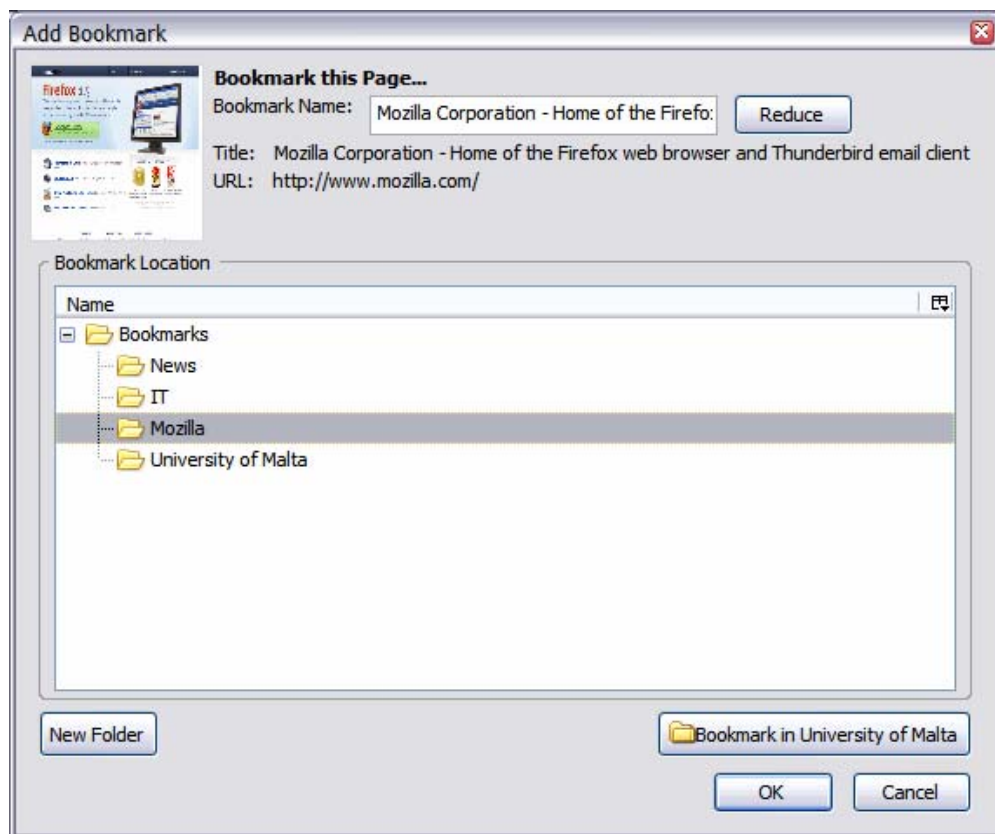


Figure 22 Add Bookmark Dialog

5.6.1 Bookmark Name Reduction Algorithm

Initially the bookmark name is set to be the page title, but in many cases the page title is rather long and contains phrases which can be removed. This algorithm enables the user to reduce a long bookmark name automatically or by a click. This

algorithm removes text which appears after the ‘-’, ‘:’ or ‘|’ characters provided that they are not in the first few characters. Many web page authors include such characters to divide their title. In the case the title starts off with the phrase “Welcome” or “Welcome to” this is also removed.

5.6.2 Adding a Category

The user can create a new category by clicking on the ‘New Folder’ button in the Add Bookmark Dialog (Figure 22). This will open up the Create New Folder Dialog shown in Figure 23. This dialog retains the same style as the Add Bookmark Dialog with a full tree which the user selects the parent folder where to put the new created folder. In case that none or selected the folder will go in the root folder.

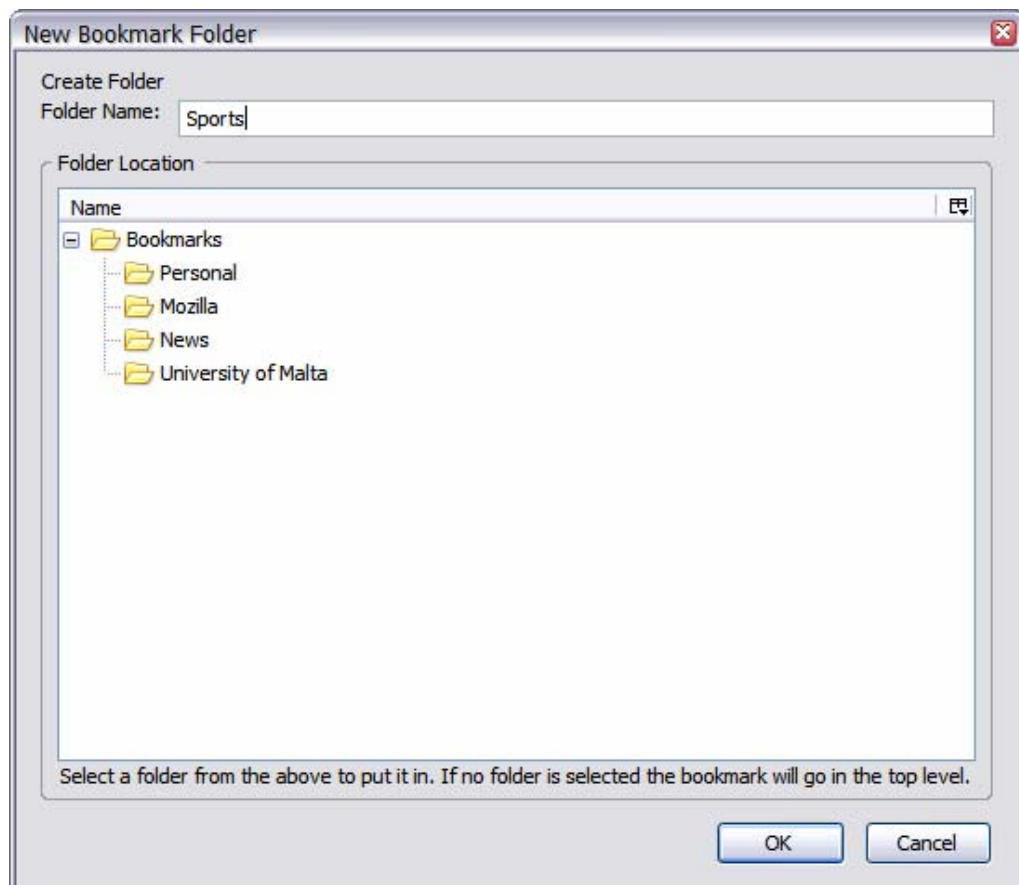


Figure 23 Create New Bookmark Category Dialog

5.7 Operations on Bookmarks

The bookmark manager (Figure 24) offers some functions to manage the bookmark folder. These include deletion of bookmarks, moving bookmarks from one category to another, changing the position within the same category, and renaming of bookmarks. All these functions manipulate the RDF data source directly.

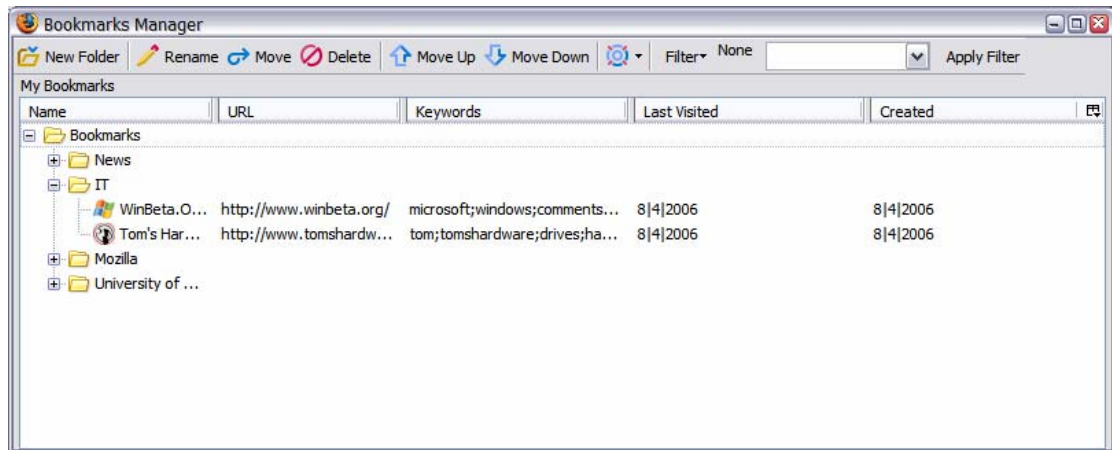


Figure 24 Bookmark Manager

Below is a list of the operations and what happens internally:

- Delete – deletes the RDF node from the data source and page from cache and thumbnail unless it's a bookmark category. In such a case all bookmarks inside of it are deleted recursively. In the case of a category the equivalent history category is also deleted.
- Rename – changes the RDF node name attribute.
- Move Up/Down – removes the RDF node from the container and inserts it at an index +1/-1 from current unless it's at position 0/end already.
- Move to Different Category – removes the RDF nodes from the container and adds it to the new category.
- Sorting – sorting is done by arranging the nodes inside of a category to be in ascending order by their name, but categories are considered as smaller than bookmarks so they will be placed at the beginning of the list.

5.7.1 Bookmark Verification

This is a utility (Figure 25) which will check if the bookmark is still active by sending an http HEAD request to the server. On response the user will be notified on the response code that was returned. A code of 200 indicates that the page is still online while other codes normally mean failure. If the page is not a dynamic page then the server returns the last-modified-date. This can be used to check if the bookmark has been visited since it was updated and notify the user if it has.

To send the HEAD request an XMLHttpRequest is used. XMLHttpRequest can either work in synchronous or asynchronous mode. In this case requests are sent asynchronously to be capable of updating the UI (activity percentage bar) while response is waited.

Although this utility can detect if a bookmark is offline it cannot make decisions on whether to delete a particular bookmark. This is due to the nature of the www. Some sites are offline for days, maybe even months but then they come online again. It's in the hand of the user to decide whether or not to delete an offline bookmark.

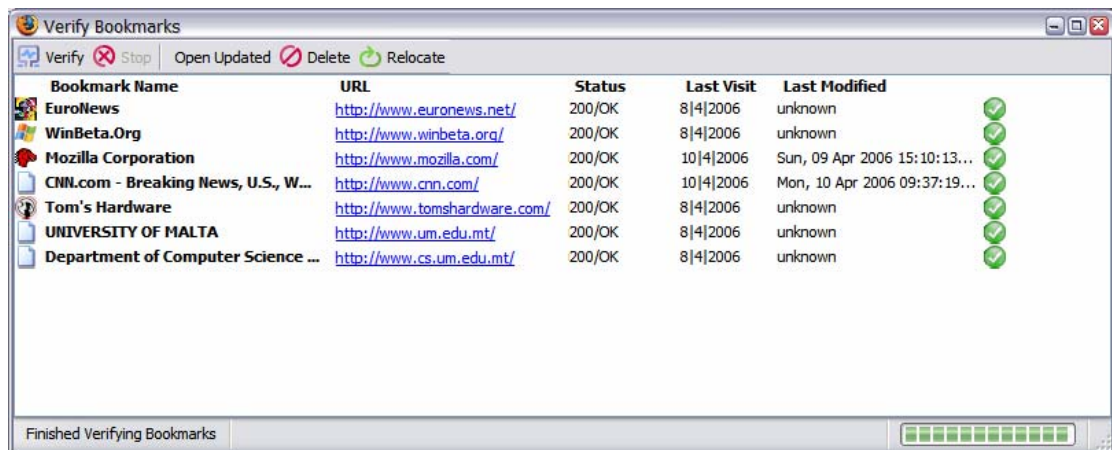


Figure 25 Bookmark Verification Utility

5.7.2 Bookmark Manager with Thumbnails

This is another bookmark manager which uses a different approach and layout than standard bookmark managers. Instead of having the bookmark represented by their

name (normally being the page title) it shows a much deeper description by using a thumbnail of the web page.

It has been shown in [16] that the title and page thumbnail (referred as minipage) is a better representation than any other representation. This is due to the fact that nowadays many webpages have their own graphic content which is unique and the user has more chances of remembering the page layout and graphical content than the name or URL. Thus this secondary bookmark manager (Figure 26) adopts this approach and a thumbnail (100pixels by 100pixels) is shown beside each bookmark.

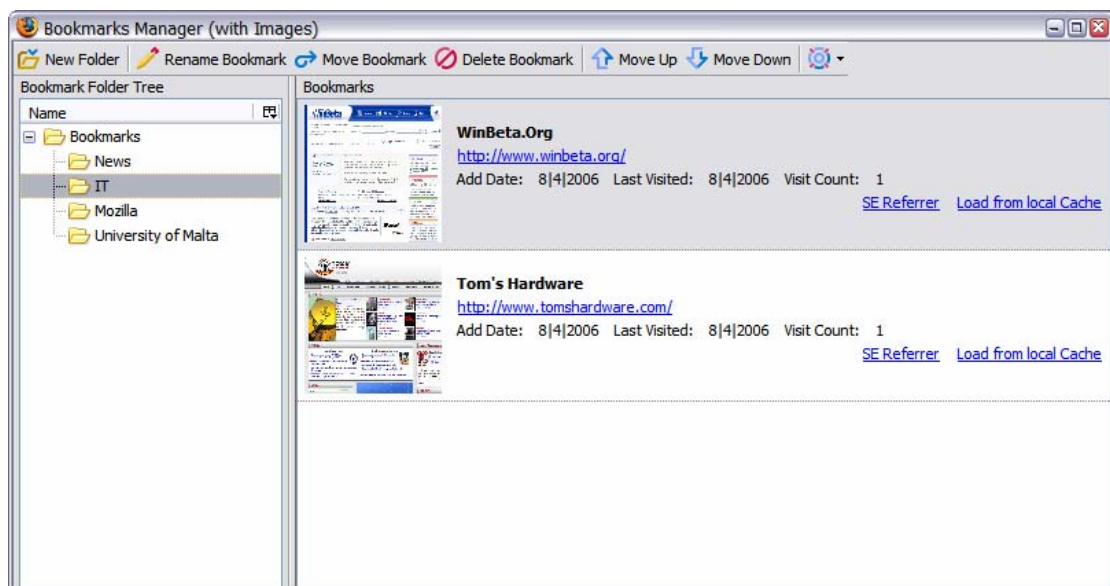


Figure 26 Bookmark Manager with Webpage Thumbnail

This thumbnail is generated by using the new html canvas features present in Firefox 1.5. It uses the same functionality Firefox uses to draw a window onto screen but obviously reducing its size. This is saved to disk in png format inside of a folder (named hyperBKIC) situated in the user's profile directory using an XPCOM component from Pearl Crescent.

5.7.3 Bookmark Import/Export

To make it easy for users to switch from other bookmark managers to this there is the option whereby bookmarks can be imported from a Netscape bookmark file. When this is done the bookmarks are inserted into the RDF data source in their appropriate

category and the equivalent history category is created. The only problem is that pure classification cannot start until all bookmarks have been visited at least once. This is due to the fact that each bookmark needs to have the set of keywords that represent it. For this reason a First Bookmark Visit Tour is available.

Similarly an export utility which exports the bookmarks from the RDF data source to a Netscape bookmark file is available. This operation is available through a wizard for simplicity. The output is either an RDF file (for backup) or else a Netscape bookmark file which is compatible with all the other browsers.

5.7.4 First Bookmark Visit Tour

This tour is to be used normally after the user imports a set of bookmarks from an external file. This tour will make it possible to visit all bookmarks that were not visited in sequence. These unvisited bookmarks will not have any keywords so once visited these will be updated. As a network enhancement images are not loaded during the tour so that the pages download faster.

5.7.5 Divide Category Wizard

When a category exceeds the number of bookmarks that it holds a special wizard is popped up. It is used to limit the number of bookmarks inside a category which defaults to 20, which is approximately the number of bookmarks that can be visible simultaneously inside a menu (without scrolling). This number can be increased/decreased through HyperBK preferences as the user wishes.

One might argue that such a feature is intrusive to the user as it will popup after bookmarking a page. Although this is true, it is the only way a bookmark category can have its size kept within bounds. But note that this wizard can be cancelled although it will popup the next time a bookmark is added to the same category. To be disabled forever the user must set from preferences the maximum number of allowed bookmarks in a category to be 0. In this case a bookmark category can have unlimited bookmarks inside of it.

The wizard operates in 3 simple steps.

1. First it will prompt the user to create a new category (Figure 27).
2. Then the category with the exceeded size will be divided either manually or automatically. In the latter the user will still be capable of doing any modifications to how the new category has been built up. The automatic division algorithm works by computing the top three keywords that are present in a category and all subsequent bookmarks that have these three keywords are kept in that category. Those that do not are moved to the new one (Figure 28)
3. Clicking on Finish will save the changes.

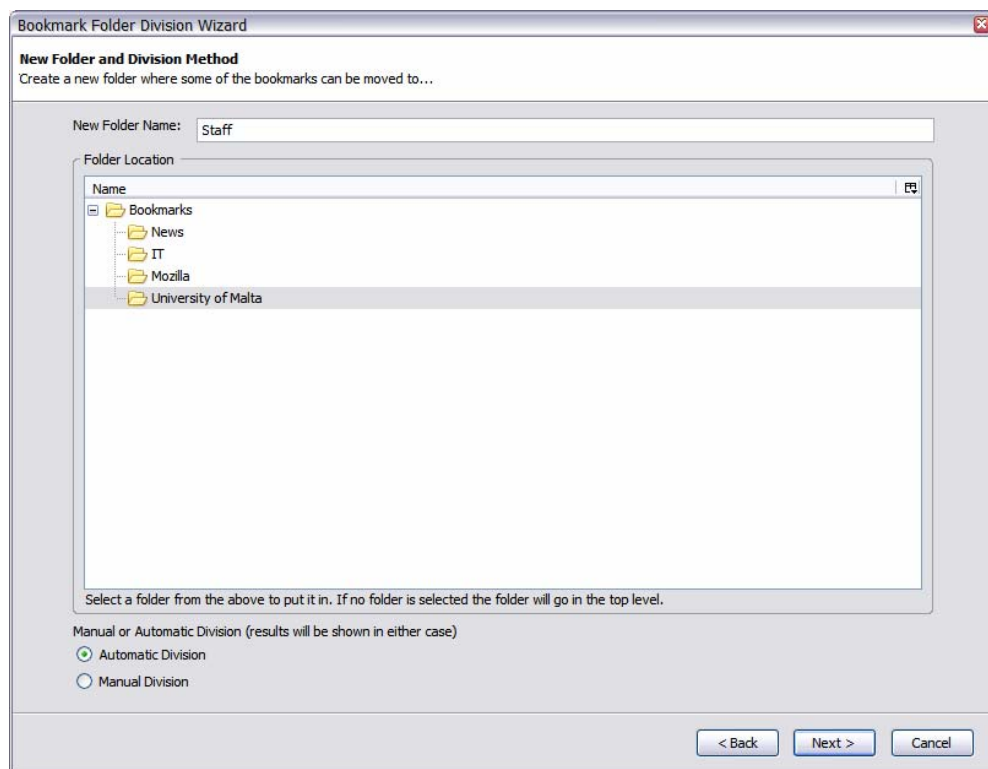


Figure 27 Limit Wizard (Create New Category)

A problem with this feature is that under the Bookmarks' Root, different categories with totally different topics are present. Thus the above algorithm will fail as no equal keywords will be found. Another problem is that some users might want to have particular webpages bookmarked under the root anyway. For these reasons the Bookmarks' Root Category is excluded from the limit wizard and can have unlimited number of bookmarks/categories.

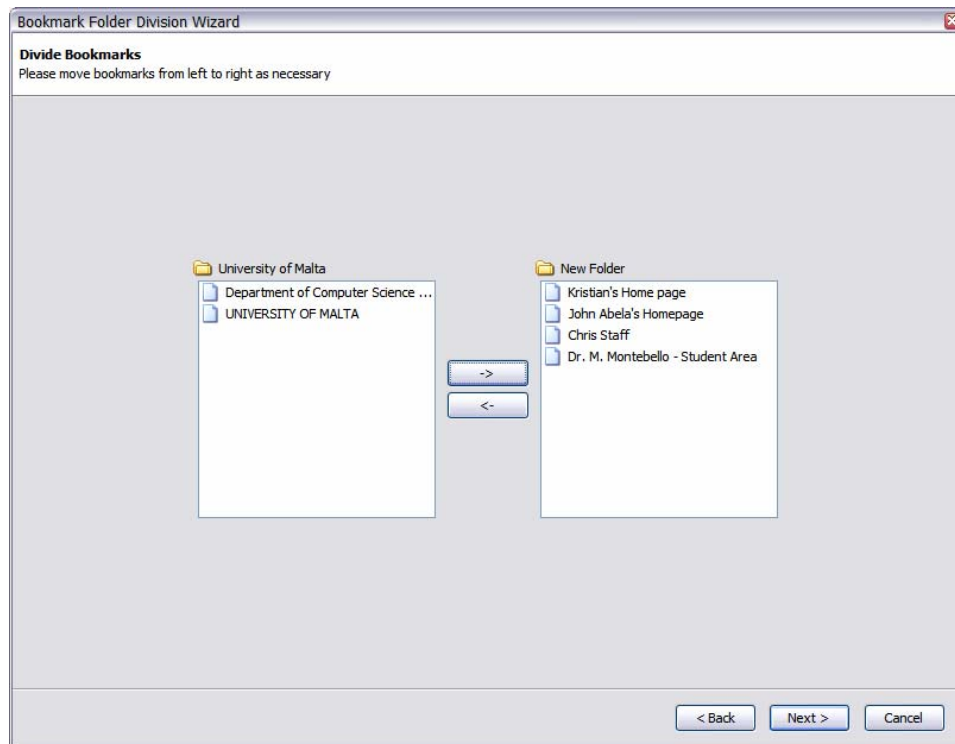


Figure 28 Limit Wizard (Divide Category)

5.7.6 Bookmark Relocation

When a bookmark fails to load HyperBK will load a copy of the webpage from local cache. This can be done as after bookmarking a page a copy of the webpage is saved in a cache for bookmarked pages situated in the user profile directory. After being loaded from cache HyperBK will generate a search query to try and relocate the document. This search query consists of the page title. The user can then manually look for a result which might match the offline document.

Perfect relocation is quite difficult, for the following reasons:

- A document changing location can also change in structure (being re-updated)
- There might exist multiple copies of a document, of different versions.
- The search engine might not have been updated still thinking that the document is online

5.8 History

As soon as a page has finished loading its URL is added to the history list. The history contains the following data about any visited page:

- Page title
- Page URL
- Page keywords (used in classification algorithm)
- Date and time last visited
- Page referrer

The categorization process is similar to that performed when a page is bookmarked. The only difference is that the page entry is added to the history not to the bookmarks and if the categorize method returns no result the node is inserted into the uncategorized container. Search pages from Google, MSN, Yahoo or any other search engine are added to the Search Pages category. This ensures that all searches are grouped under one single category.

A particular URL can only be present once inside of the history and child of only one category. Once a page has been added to a particular category it stays in it, even if a newer category is created and this matches more than that particular page. This is so for performance reasons as it would be too time consuming to rearrange the whole history every time a page loads. Although this is expensive it can be easily implemented if required as the page keywords are stored with each history entry. A page may only change category if it is re-visited and a better match is found. Unfortunately this leads to a problem: if a user revisits a webpage by clicking on it from a category in history viewer window (Figure 29) and when the webpage loads a better category is found then this would disappear from that category and appear in another, becoming lost from user's sight.

As an example take that Page X visited some time ago is situated in Category A. While viewing the history list from the History Viewer Window (Figure 29) the user decides to open Page X (by clicking on it). In this particular case as soon as Page X loads, the algorithm finds Category B which suits more Page X, so Page X is moved

to Category B. The problem is that as the page moves the page entry in the history viewer window moves too and it won't be visible, unless the Category B is expanded.

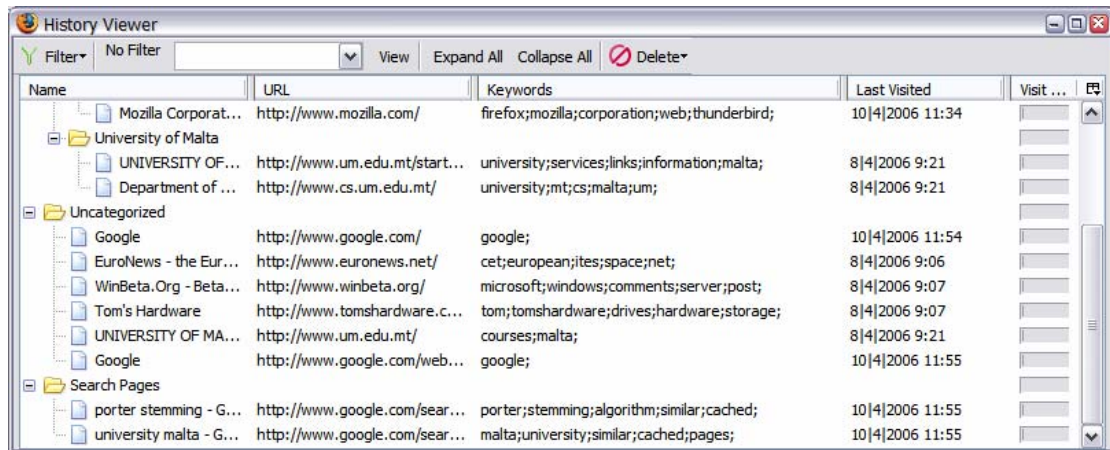


Figure 29 History Viewer

5.8.1 History Views

Although the history is already divided up into the bookmark categories, the use of filters to filter up the number of history entries is still required. The filters that are available to the user are:

- By Date
- By Section
- Visit Count
- Keyword
- By Same Visit Period

The history still maintains the structure of the bookmarks hierarchy; the only difference when one of the above filters is applied is that the history entries are reduced to match the filter parameter. Each of these filters is applied by traversing all the nodes in the history data source and selecting those that match the filter parameter. Containers are never excluded from the results except in the 'By Section' filter which would display only one category.

The 'By Date' filter requires its parameter to be a date. Only pages that were visited on that date will be displayed. The only problem with such a filter is that as the history

only stores the last visit time and date so if a page is visited more than once the most recent visit will override the previous one. This obviously leads to loss of important information when the user might want to see pages that s/he visited in the same period before the last visit. This approach was adopted as browsers (Firefox and IE) use this method of storing the history list. In fact this approach reduces drastically the number of history entries as 58% of pages visited are revisits [8]. If this approach was not adopted then in my opinion the history list would be too large and thus much more complicated to traverse.

The 'By Section' filter will filter out all categories except the selected one thus the user can reduce the number of nodes on screen to be able to traverse this category more easily.

The 'Visit Count' filter will show all pages that were visited more than the visited count parameter.

The 'Keyword' filter will show all pages that match the keyword entered with the page title, page URL, or keywords that represent that particular page.

The 'Same Visit Period' is a different kind of filter. It will be applied to a particular page and will show all pages that were visited in the same time span as that page was visited. The timeframe taken is that of two hours before and two hours after by default but this value can be changed from the preferences in "Same Period" parameter (see section 5.12). This is particularly ideal if a user remembers a page that s/he visited but does not remember the page s/he requires but just knows that they were visited in the same period. Such a filter should pick up such a page.

5.9 SE Referrer

In many cases a user starts off his browsing sessions from a search engine. The search query that a user uses to search for a particular page is a good indicator of what the user is looking for especially if one of the results is bookmarked. So with each bookmark the so called SE Referrer is stored. This is the search engine results

page URL that was used to find the page. For this reason when the user decides to bookmark a page the URLs in history that follow that trail must be checked to see if any match a particular search engines' result page.

For this reason to allow flexibility the Firefox preference `extensions.hyper.seurl` holds part of the search engine search URL together with the query variable in the following format:

```
searchengineurl,variable;  
searchengine.com/search,p;searchengine2.com/results,q;
```

The above encoded string would match URLs like the two below:

```
www.searchengine.com/search?p=cats  
www.searchengine2.com/results.aspx?q=cats
```

These URLs would refer to two search engines' pages URLs with a search query on the term 'cats'.

As a default value this preference is given the following string:

```
google.com/search,q;yahoo.com/search,p;msn.com/results.aspx,q;
```

This should match Google search page, Yahoo search page and MSN Search results page. The user can easily modify in case the search engine changes the URL or add other search pages URLs to satisfy his needs.

To find the SE Referrer of a bookmark the history is used. This is possible because each history entry consists of the webpage URL together with the referrer URL that lead to that webpage. So when a webpage is added to the bookmark list, its referrer is checked to see if this matches a search engine's search page URL. If not then the referrer of this page is checked and so on till a valid one is found or else a null referrer which would indicate that the browsing started off by following a bookmark, link from history or manually entered URL.

Firefox maintains the correct referrer when any link is clicked. In fact when a new link is opened the calling function takes as a parameter the link URL and the URL of the referrer. This means that the referrer is always valid for links in web pages even if

they are opened in a new window, new tab, or the same existing tab. Thus if a referrer is valid the SE referrer can be located (if it exists).

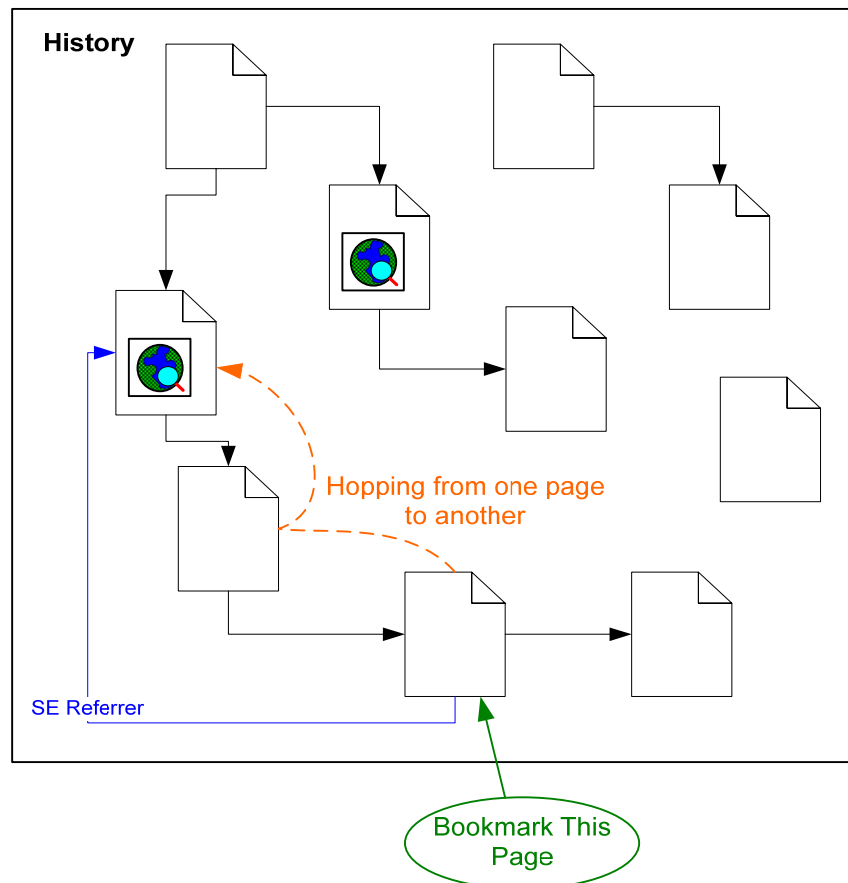


Figure 30 SE Referrer Finding

5.10 See Also

The 'See Also' feature is used to fetch a set of similar pages from a search engine. The two different methods that are used to compute the query for the See Also are:

- By using the SE Referrer
- From the top page keywords

5.10.1 Using the SE Referrer for 'See Also'

The SE Referrer is particularly useful to search for similar web pages that can fall inside of a particular category. By grouping all the terms from the queries found in all SE Referrers that are present inside a particular category, the top used terms are

then used to construct a query which will be sent using SOAP to Google. Google will conduct the search and produces a set of results similar to those present already in the category.

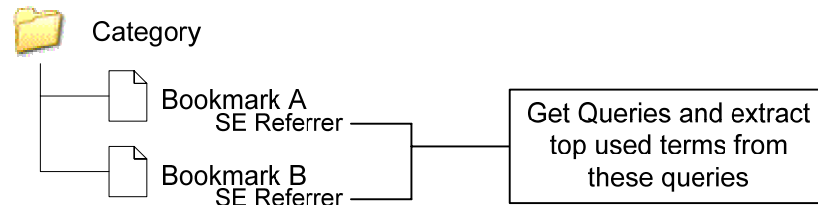


Figure 31 See Also (using SE Referrer)

5.10.2 Using the top Keywords for 'See Also'

A different approach for constructing the query that is used to find similar pages to a particular category is to use the keywords that are extracted from the pages automatically. These are the top five keywords that are used in the classification algorithm. By grouping them and finding the most common a search query can be constructed. In my opinion there is a problem with this query, as from the couple of examples tried out the results were very poor. For this reason a different approach was taken, instead of using the standard Search over the WWW the search was done inside Google's Directory. In this case the results were satisfactory. (See evaluation section 6.3.3)

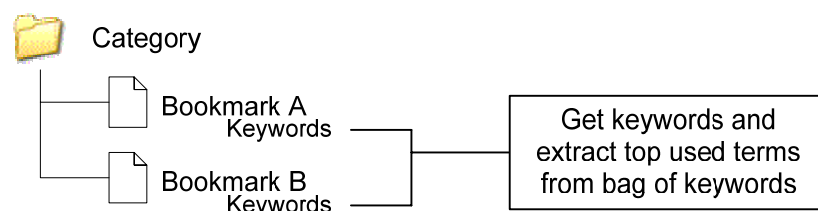


Figure 32 See Also (using Automatic keywords)

5.11 In Page Suggestion

This feature is used to indicate to the user the links found inside of a page that are already bookmarked. Each hyperlink inside of the page if bookmarked will show a tooltip saying "HyperBK Already Bookmarked Link". This tooltip approach is taken not to disrupt the page contents like it would happen if an arrow is inserted or change of link colour. Moreover if link colouring was adopted there is the problem that recently

visited links (which are also bookmarked) would have their colour changed overriding the visited hyperlink colours the browser sets. This is due to the fact that Firefox does not change the html but changes the recently visited link colour internally.

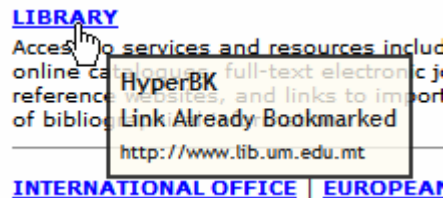


Figure 33 Tooltip over bookmarked link

This feature is extended to pages on the same path (i.e. where a domain match occurs). On page load the list of anchor tags is traversed and each one of them has its `href` attribute compared against the bookmarks' URLs. If a match is found then a message "Link Already Bookmarked" (Figure 33) is added to the tooltip, while if the domain matches but not the path then the message "Other link on same path already bookmarked" (Figure 34). In the case there is an existing tooltip this is appended to the bottom of this tooltip below the `<hr>` (horizontal line) as in Figure 34.

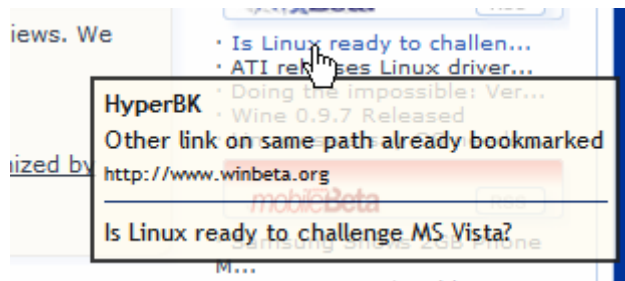


Figure 34 Tooltip over link with same path as bookmark and link title shown underneath

To show this tooltip instead of the standard custom title attribute a new attribute was added (i.e. `htitle`). The tooltip is a custom html element which is added inside of a script tag on page load. This feature is controlled by two options in preferences (ref Section 5.12). These two preferences can turn off this feature, the first option will only offer a tooltip for URLs that match a page while "On Same Path too" will result in a tooltip where the domain matches but not the path.

5.12 HyperBK Preferences

For easy configurability a number of preferences are available to the user so that s/he can modify the extension behaviour. A special window is used for this purpose (see Figure 35). The preferences that can be modified are shown in Table 4:

Table 4 HyperBK User Preferences

<i>Preference</i>	<i>Type</i>	<i>Default Value</i>
Allow to rebookmark	boolean	true
Automatically reduce bookmark name	boolean	true
Maximum number of bookmarks in category	int	30
Maximum number of fast bookmarks	int	30
Maximum number of days in history	int	20
Same Period (hours)	int	2
Offer tooltip suggestions	boolean	true
Offer tooltip suggestions for same path	boolean	true
See also Algorithm	int	2
Google Developer Key	string	null
Search Engine URLs	string	
Hide original Bookmarks Menu	boolean	true
Show Fast Bookmarks Menu on menu bar	boolean	false
Maximum Script Running Time	int	50

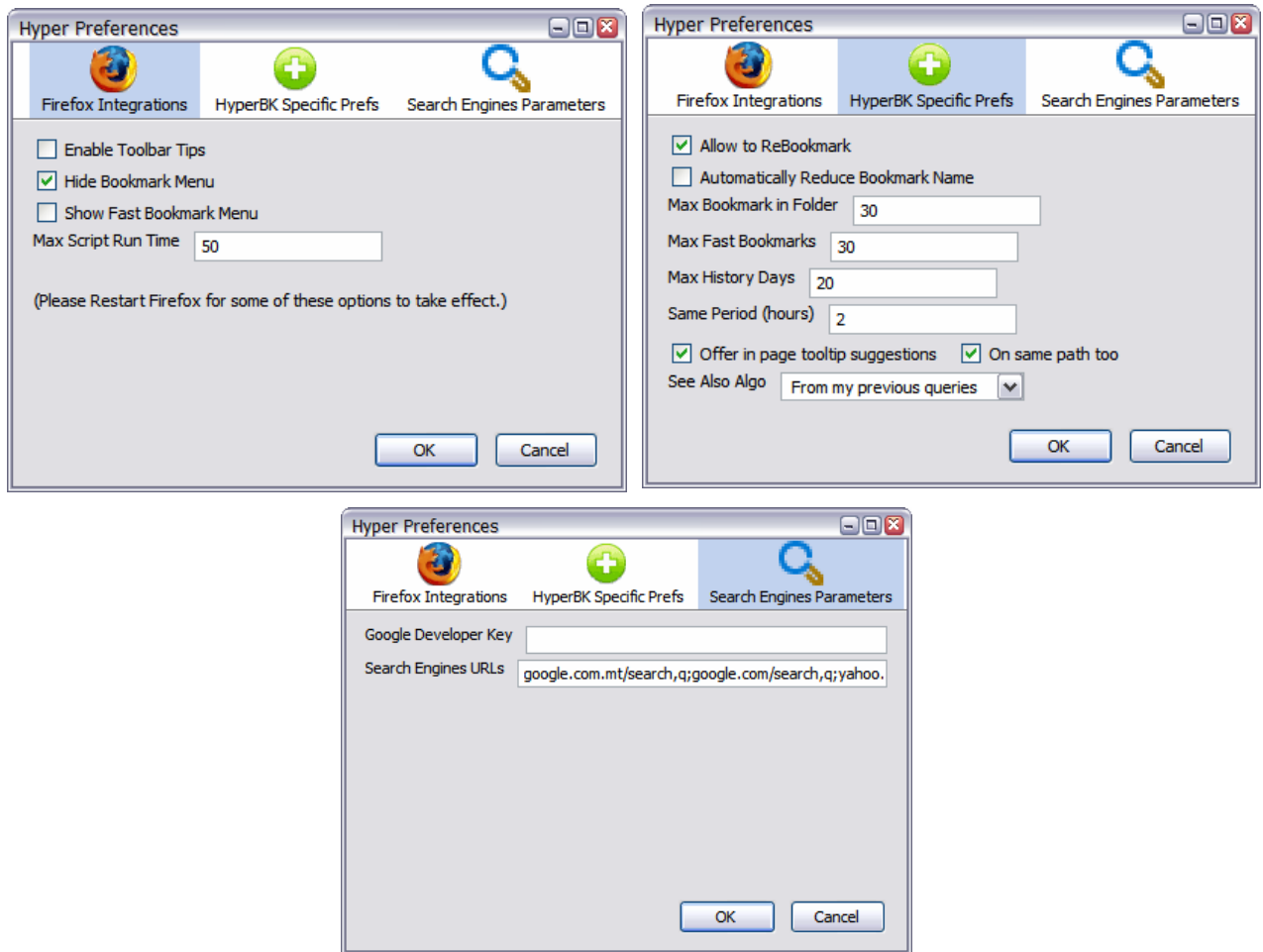


Figure 35 HyperBK Preference Windows

5.13 Summary

This chapter explained how HyperBK has been implemented for Mozilla Firefox which includes the bookmark functionality, classification algorithm and history management. It also described the SE Referrer, See Also and other features such as the divide category wizard which is used to maintain the size of a category within bounds.

Chapter 6: Testing and Evaluation

6.1 Introduction

This chapter consists of two parts; the first section is a test report on HyperBK, the list of test conducted and the results obtained. The second part is an evaluation of HyperBK and of how much use it is. Following this there is a list of possible solutions to some problems that were encountered.

6.2 Testing

When HyperBK's implementation was completed a number of tests were performed to ensure that the extension was working as specified. All tests were performed on Firefox version 1.5.0.2.

Testing was divided into four parts:

- Testing overall Extension Integration
- Testing Bookmark Manager
- Testing History Manager
- Testing Classification Algorithm

The biggest problem with JavaScript code is that because it is interpreted and not compiled some syntax errors may still be present in a piece of code even though the script file would have been loaded and parsed correctly. The types of errors that can occur are due to the late binding as method names are not checked on load. This actually means that to ensure that there are no such errors each code block has to be executed.

The first series of tests involve the extension's integration with Firefox and such tests were performed on three different platforms Windows, Linux and MacOS. This was done in order to ensure that HyperBK could really run on these different platforms.

Such tests involved checking that the extension installs correctly, and that the files/cache directories are created in the proper location (user profile).

The next series of tests involved the Bookmark/History Manager; they ensured that the bookmarks/history was being updated correctly in their respective data sources. The operations carried out were checked to see that the results were correct such as deletion of bookmarks, update and so on.

The final series of tests were regarding the classification algorithm. This was merely a test to check that there were no errors in the code and not that the classifying algorithm was correctly classifying the documents. The evaluation described later on in this chapter was done to check that aspect of HyperBK.

Table 5 List of Tests Performed

<i>Test No.</i>	<i>Test Name</i>	<i>Platform</i>
01	Correct Extension Deployment	All
02	Proper overlay integration	All
03a	Add Bookmark I	All
03b	Add Bookmark II	Windows
04	Bookmark Tour	Windows
05	Bookmark Manager	Windows
06	Bookmark Manager Filters	Windows
07	Bookmark Verification	Windows
08	Bookmark Import/Export	Windows
09	Bookmark load Cache/Relocation	Windows/Linux
10	See Also	Windows
11	Page Addition to History	Window/Linux
12	History Manager	Windows
13	History Manager Filters	Windows
14	Browser Sidebar	Windows
15	Classification Algorithm	Windows
16	SE Referrer	Windows

6.2.1 Detailed list of Tests Performed

Test:01 - <i>Correct Extension Deployment</i>	Platforms: All
Description: Check that deployment of xpi file is correct. This includes: <ol style="list-style-type: none">1. XUL documents, components and preferences2. Verify that hyperBKIC and hyperCache folders are created in the user's profile directory.3. Verify that the 3 RDF files: hyperHist.rdf, hyperFastBK.rdf, hyperBookmarks.rdf are created in the user's profile directory.	
Result: Pass	
Known Issues: On Linux Fedora the component from PearlCrescent failed to install. This should be solved in the next update of the component.	

Test:02 - <i>Proper overlay integration</i>	Platforms: All
Description: Check that the extension's main overlay correctly merges with Firefox (browser.xul). Bookmark Menu should be hidden, Toolbar and Sidebar should be added to menu and visible on selection, New Bookmark Menu and Fast Bookmarks Menu.	
Result: Pass	

Test:03a - <i>Add Bookmark I</i>	Platforms: All
Description: Verify ctrl-D and Add Bookmark Dialog, bookmark addition to bookmark file and links work correctly, from Bookmark Managers, Menu and Sidebar.	
Result: Pass	
Known Issue: ctrl-D failed on MacOS as key sequence is different	

Test:03b – <i>Add Bookmark II</i>	Platforms: Windows
Description: As the add bookmark is not just adding a bookmark URL to the bookmark file then testing has to be carried out in more depth. This things to check include: <ul style="list-style-type: none">• Adding a bookmark after following a series of links.	

<ul style="list-style-type: none"> • Adding a bookmark after following links from Search Engine results page. • Adding a bookmark to a category where it is already present. • Adding a bookmark to a category when the bookmark is already present in a different category.
Result: Pass

Test:04 - <i>Bookmark Tour</i>	Platform: Windows
Description: Test that bookmark tour moves from one bookmark to the next as expected. Starting tour from both sidebar and menu. In the case of the sidebar start test on category or in the middle of a category.	
Result: Pass	

Test:05 - <i>Bookmark Manager</i>	Platforms: Windows
Description: Verify Bookmark operations such as Move, Delete, Rename and Create New Category. Inspect the RDF file to check the results.	
Result: Pass	

Test:06 - <i>Bookmark Manager Filters</i>	Platform: Windows
Description: Verify that filters work correctly on the bookmarks available according to the specified parameters.	
Result: Pass	

Test:07 - <i>Bookmark Verification</i>	Platform: Windows
<p>Description: Verify Bookmark verification utility with different bookmarks. To ensure proper functionality the bookmarks need to be with different response codes, and having some offline bookmarks too.</p> <p>Test must include:</p> <ul style="list-style-type: none">• Starting of verification• Stop in middle of verification• Open Updated• Delete of Bookmark	

<ul style="list-style-type: none"> Relocate of Bookmark
Result: Pass

Test:08 - <i>Bookmark Import/Export</i>	Platform: Windows
Description: Check that Import from Netscape Bookmark File works correctly and export to Netscape Bookmark File creates equivalent structure. The import has to be performed on both an html file exported from Microsoft Internet Explorer and another from Mozilla Firefox.	
Result: Pass	

Test:09 - <i>Bookmark load Cache/Relocation</i>	Platform: Windows, Linux
Description: This test should be done on a bookmark that is offline to verify that the extension properly loads the local copy from cache and then generates the re-location query and sends the search request.	
Result: Pass	

Test:10 - <i>See Also</i>	Platform: Windows
Description: Verify that See Also Algorithm returns top keywords with two methods: SE Referrer and automatic generated query. The method used should be equivalent to the one set in HyperBK Preferences. In the case that the user has set to use the SE Referrer, if no SE Referrers are present than the default algorithm should be used.	
Result: Pass	

Test:11 - <i>Page addition to History</i>	Platform: Windows, Linux
Description: Check that page observer will add/update page to history. The page entry should also be ensured to enter the correct category i.e. the same category as if it were bookmarked.	
Result: Pass	

Test:12 - History Manager	Platform: Windows
Description: Check that proper URL opens on double click and delete functions work correctly. Also check the Highlight Referrer option which should highlight the correct referrer of a page.	
Result: Pass	
Known Issues: <ol style="list-style-type: none"> 1. As each entry in history is kept only once then if a page is visited twice and this page was a referrer of another page then the page flow of the previous session would be lost. 2. Collapse All won't collapse whole tree. This was done so as top root node needs to be visible at all times. 	

Test:13 - History Manager Filters	Platform: Windows
Description: Check that each filter picks up the nodes from history as entered in the filter parameter.	
Each filter should be checked with the following values	
<ul style="list-style-type: none"> • By Date – today and any previous day • By Section – Uncategorized and any History Category • Visit Count – A value out of the drop down and any other value • By Keyword – part of a URL, part of page title and keyword • By Same Visit period – needs to be tested on bookmarks from different days. 	
Result: Pass	

Test:14 - Browser Sidebar	Platform: Windows
Description: Check that sidebar search functions work correctly, and correct pages are opened on double click.	
The sort, delete, and rename functions have been performed on a bookmark and a category.	
Result: Pass	

Test:15 - <i>Classification Algorithm</i>	Platform: Windows
Description: Ensure that correct keywords are extracted from the html document and classification algorithm works as specified. Best way to ensure that it is working as it should is to run it in debug mode with watches on some of the most important objects to ensure proper execution.	
Result: Pass	

Test:16 - <i>SE Referrer</i>	Platform: Windows
Description: Ensure that correct SE Referrer is retrieved and stored for a bookmarked page. This has to be tested on a page bookmarked just after a search page and also after following a couple of links.	
Result: Pass	

6.2.2 Tests Summary

The above tests ensured proper deployment of HyperBK and correct functionality (addition, deletion and maintenance) of the bookmark/history datasources. These tests also ensured that the UI's layout displays correctly and in the correct format. The tests also ensured that the html pages are properly parsed and that the content is correctly extracted out of them. Basically all aspects of HyperBK were tested.

6.3 Evaluation

This section will give an evaluation of the system together with the problems that were discovered after this evaluation was carried out. This evaluation process was performed to test HyperBK functionality and get feedback of how the users will find it of use. Although the tests in the previous section ensured that the code worked as expected according to the test performed this is not enough to ensure that HyperBK is of use.

6.3.1 Classification Algorithm Evaluation

The best way to evaluate the classification algorithm is to use real data. In [17] the evaluation of the FMM (Finite Mixture Model) was carried out by selecting a subset of

Reuters Newswire distribution of documents from different categories and using FMM to classify them. The resultant classification was then compared with the original classification, where a match would indicate correct classification.

In the case of evaluating the classification algorithm in HyperBK the evaluation was carried out on bookmark files from real users. The resultant classification is then matched with the one the user had done manually and if both methods classified a bookmark into the same category this would mean that the classification algorithm detected the best match. Although this is true there is no way of checking that the bookmark was bookmarked by the user into the best matching category in the first place.

As bookmarks are personal, the submission of bookmarks had to be done in an anonymous way in order to maintain user privacy. For this reason a web portal was created where the user could upload a bookmark file and get in return a numeric ID to identify him throughout the whole evaluation process.

As the classification algorithm works in an incremental fashion it is not possible to categorize a whole bookmark file automatically at one go. In fact the algorithm needs to have some bookmarks that are already present inside a particular category to match a new page with that category. For this reason the procedure below was adopted in order to get fair results on the classification:

1. We created a training set and a test set by removing 10 bookmarks out of the bookmark file at random. The removed bookmarks were kept in a separate file for later use, including the parent category they were originally classified in.
2. The resultant bookmark file was imported into Firefox.
3. All bookmarks were visited once (using the First Bookmark Tour, refer to section 5.7.4).
4. The bookmarks that were removed in step 1 were then visited in sequence and each one was bookmarked to the original bookmark category. The suggested category is noted.

An alternative method to the above is to shuffle the bookmarks in the bookmark file randomly and visit one bookmark at a time adding it to the proper category. Then one can determine the amount of bookmarks a category requires to start matching other similar bookmarks.

6.3.2 Classification Results Obtained

Due to the lengthy operation that is required for this evaluation as each bookmark would have to be visited once, the operation was carried out on eight bookmark files out of the thirty bookmark files collected. These bookmark files were collected from all four years of B.Sc. IT students. The eight bookmark files selected were files which were organized in some manner as many bookmark files submitted were just a collection of bookmarks without any type of organization.

Table 6 Classification Evaluation Results									
ID	BKs	Cat.	Root BKs	Root Cat.	Hits	Misses	Near Hits	Approx Precision	Precision
23740	425	105	49	33	7	2	1	0.80	0.70
24166	330	64	2	26	8	0	2	1.00	0.80
88014	240	53	21	12	7	2	1	0.80	0.70
23248	197	45	6	9	5	3	2	0.70	0.50
79231	158	38	18	18	4	3	0	0.57	0.57
58917	139	29	21	11	8	2	0	0.80	0.80
76243	38	11	3	11	5	0	0	1.00	1.00
80999	22	7	1	5	4	0	0	1.00	1.00
Totals					48	12	6	6.67	6.07
Averages					4.8	1.2	0.6	0.67	0.61

Legend: *BKs*: Total Bookmarks, *Cat.*: Total Categories, *Hits*: bookmarks allocated into correct category, *Misses*: bookmarks allocated wrongly, *Near Hits*: when allocated to parent category (excluding in case off Bookmarks Root), *Approx Precision*: near hits + hits / total, *Precision*: hits / total (refer to Figure 36).

Table 6 shows the results that were obtained. The amount of bookmarks tested for each file ranged from ten to less in case that the bookmark file did not contain a sufficient number of categories from which to take the bookmarks. The bookmarks were selected from categories which contained a substantial amount of bookmarks (more than five) in order for the classification algorithm to work.

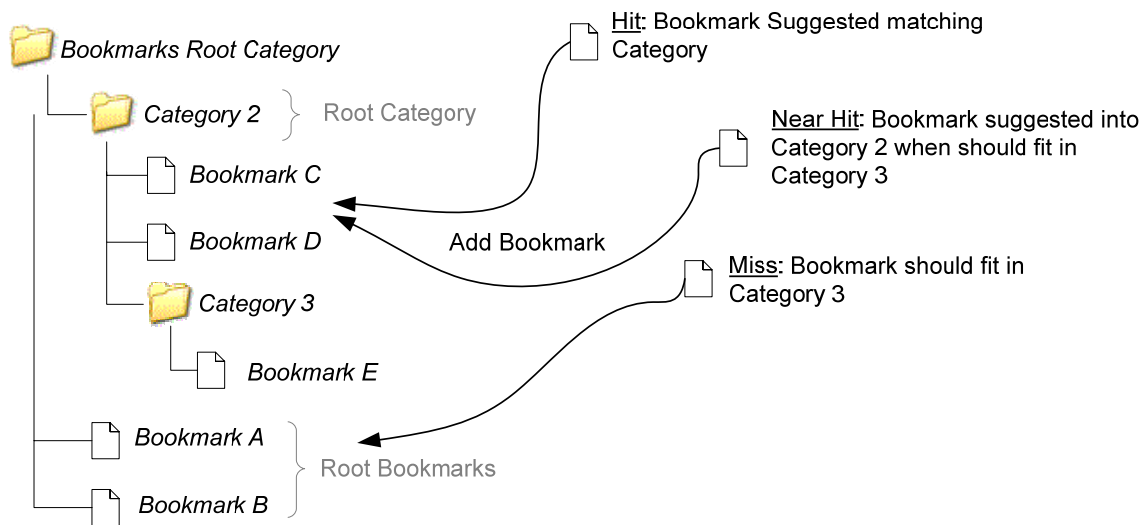


Figure 36 Categorisation Evaluation Hits

As one notices the algorithm is much more precise on average sized bookmark files. The precision is also dependent on the way the user had sorted the bookmark file originally. If the user adopts an approach which is foreign to the page contents (as explained in section 6.4.2) then there is a higher chance that a page will be allocated to a different category.

There is the possibility that when these bookmarks were classified originally by the user as there was no hinting to a category the user may have bookmarked some of them haphazardly. If there was hinting they may have been allocated to a different category.

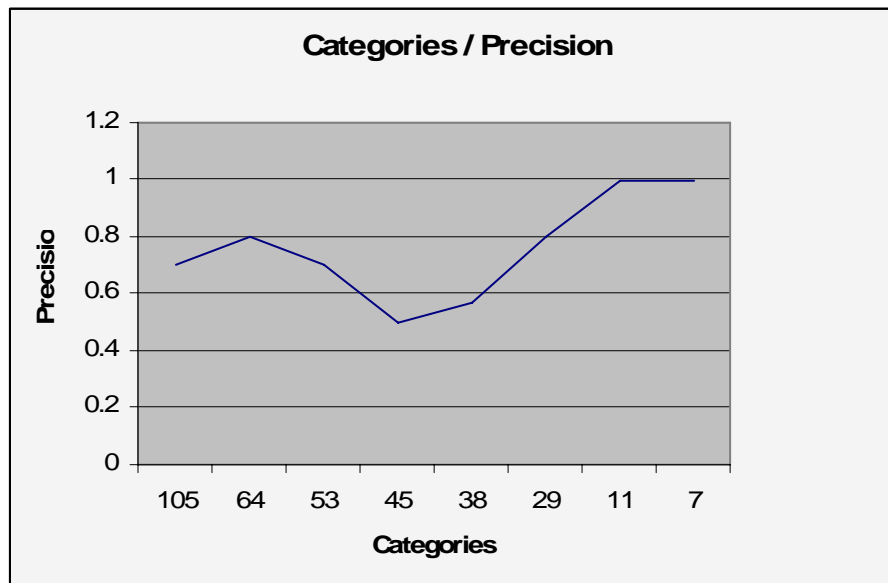


Figure 37 Number of Categories against Precision Obtained

The figure above (Figure 37) is a graphical representation of the Categories plotted against precision. One notices that the less the categories that are present the higher the precision. The dip observed in the middle of the graph is probably due to the organization of bookmark files (ID: 79231 & 23248). Bookmark file 79231 lacks structured organization while bookmark file 23248 has multiple categories on the same topic (hence the two near misses).

(Refer to Appendix B for more details on how this evaluation was carried out)

6.3.3 See Also Evaluation

The 'See Also' feature is an option whereby the user gets a list of see also links for a particular bookmark category (ref section 5.10). There are two methods how the See Also query is computed: from the automatically collected keywords and from previous search queries using SE Referrer (ref. section 5.10). Unfortunately the second method cannot be evaluated at this time because for this to be used there needs to be at least one SE Referrer present in the category. This can only happen if the user would have bookmarked the page after visiting a search engine result's page. A number of users would need to be recruited to use Firefox with the HyperBK extension over a period of time and this was not possible due to time constraints thus only the first method will be evaluated.

The procedure for evaluation is quite simple; some See Also queries were generated from categories taken at random. The results were submitted to the bookmark file's original user. The bookmarks files used were those that were used in the classification evaluation. Each user had to rate the search results given for the particular category from one of the following: Useless, Good, Really Good, Prefer to do my own Search. 'Prefer to do my own Search' refers to when the query generated is not exact and some terms could be added/removed.

Table 7 shows the results obtained for this evaluation. In most of cases the score was 'Good' or 'Really Good' (41% each). Only in one case a 'See Also' query was determined to be 'Useless'. In 12.5% of the cases the 'See Also' query had to be modified which proved why it is important that the query is editable so that the user can modify it.

Table 7 'See Also' Evaluation Results

ID	Categories	Useless	Good	Really Good	Prefer my own search
23248	5	0	3	1	1
23740	3	0	2	1	0
24166	4	0	0	4	0
58917	3	0	1	1	1
76243	3	1	2	0	0
80999	3	0	1	1	1
88014	3	0	1	2	0
Totals:	24	1	10	10	3

(Refer to Appendix B for more details on how this evaluation was carried out)

6.3.4 UI Evaluation

The only way to verify how user-friendly and easy it is to use HyperBK is to carry out an evaluation. A typical evaluation in this case will consist of having a number of volunteers using HyperBK for some time (possibly a week or even more). Following this a series of questions can be asked to these users on the UI functionality of HyperBK. As browsers are used by different kinds of users with diverse skills and knowledge the volunteers have to match this distribution for precise results.

Unfortunately such a process takes quite some time to be completed as to find volunteers, distribute the extension and gather feedback back from each participant.

For this reason it was not possible to conduct this evaluation due to the limited time available to complete this project.

6.4 Classification and Bookmark Category Suggestion Problems

This section will describe some problems relating the bookmarks classification to a category and some solutions to them. It also includes ways how the bookmark categories that match could be suggested.

6.4.1 Pages with no text

As the content on the WWW can be in any format some web page authors decide to use image files or flash animations to display textual content. In the case of image content this problem is even more visible when the img tags have no alt text as while parsing the page alt attributes are examined (ref section 5.3). As no textual content is present such pages cannot be categorized. A solution would be to visit some of the children that the page has, categorize such pages and get an indication to where the page should fit in. This obviously leads to the problem of which children to choose and obviously it might be too expensive to visit all of them. But on the other hand many pages of this kind would only contain few children. Such pages are in many cases home pages with links like “Welcome to this site” or “Click to enter site”. Visiting such a link might be enough to correctly categorize the page.

6.4.2 Different Categories for the same topic

In some cases a user may decide to have two or more categories on some particular topic. The criteria for addition to one category and not the other would not be based on the content of the web pages but on some other external factor. This external factor could be time or research project. As the web pages in such categories are similar it would be difficult to determine how to automatically categorize similar web pages. The solution to such a problem is not simple.

A partial solution to the above problem could be to take into consideration the time a category was updated. If two categories exist on the same topic but one category has

been updated recently then this could mean that this category should have a higher score than the other one.

Another approach to this kind of problem would be to give options to the user where the bookmark can be inserted. Figure 38 shows this idea, where in this case an extra tab is added where the matching categories are displayed in bullet form. The user still needs to have the option to bookmark in any place s/he wants by viewing the whole tree but on the other hand the matching category tab will list all categories that match. A problem that this dialog might introduce is that it's much more complex and the user might have to switch tabs as the required category is not listed. Another problem with bullet form categories is that the hierarchal view of the categories is lost, unless the bullets contain the parents in the form: (root...parent...child).



Figure 38 Example Add Bookmark Dialog with Tabs

A better solution to the above suggestion is to show only one tree and include an arrow or bullet with the categories that match (see Figure 39). The green arrow would be on the side of the categories that match and the user can select the desired one. If the user does not like any of the suggested categories s/he can still select one of the others. This approach would make the dialog much simpler as no tabs would be present and might give an indication to the user which is the most appropriate category that the bookmark should be placed in.

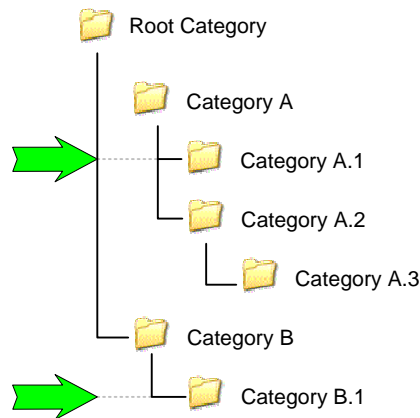


Figure 39 Bookmark Category Tree with Arrows

6.4.3 Better Classification

A way of improving the classification algorithm could be that of using the knowledge of the pages that link to and out of a particular page. The reasons for not using the children might be obvious as in many cases these would not have been visited yet and it would be too time consuming to download and process them. But on the other hand the parent would already have been processed. This can give a better indication on the category the page should fit in. It can also give a better indication on where the parent should be in the case it was classified in the wrong category due to misleading or incomplete information.

This approach can be easily implemented in HyperBK without the need of major changes. It can be done by fetching the referrer of a page in history and checking the category that this referrer was allocated to. This category would then be compared with the categories that are matching the new page. Search engine URLs would have to be filtered out in the same way as it was done for classification in 'Search Pages' category in history (ref section 5.9).

In most cases the parent and child would be members of the same category. But sometimes there might be a topic change. For this reason there needs to be detection when this topic change happens, but for this one can rely on the page keywords. This solution can solve the problem when a page has no textual content

as described above (see section 6.4.1). If a page with no textual content is visited and then the one of its children (which has textual content) is visited the chance that both pages are on the same topic is high, thus using the approach the parent would be then reallocated to the correct category.

6.5 Bookmark Related Problems

Bookmarks stored in the bookmark file are kept unique by assuming that each page has only one URL referring to it. As it is explained in this section URLs are not always unique. There can be a many-to-one relationship between URLs and a page, in that a single page can be referred to by several URLs. This can lead to problems such as the bookmark file not being correctly updated or duplicate bookmarks existing in the bookmark file.

6.5.1 Directory Index

Each web page on the WWW has a unique URL which can be used to uniquely identify that page. But web servers on the other hand would serve the default page when a directory is requested. This is known as the DirectoryIndex page which would be sent by default if a directory is requested. In such a case the URLs *www.foo.com* and *www.foo.com/index.html*, although different URLs, refer to the same page as the web server at *foo.com* would send *index.html* whenever a directory is requested. Thus if a user bookmarks *www.foo.com/index.html* but then visits the page by entering the URL *www.foo.com* there is no indication that the user visited a bookmarked page. Unfortunately the http protocol has no features whereby one can get the web page document name and so a solution to this problem does not exist.

The same situation occurs with different server names such as *www.foo.com* or *foo.com*. Both URLs can lead to the same page.

6.5.2 Bookmark URLs with Attribute Value pairs

A similar problem to the above is due to the attribute value pairs that a URL can contain. A simple approach can be that of excluding these attribute value pairs, but

many dynamic webpage's content is bound to a value held in one of these attributes. Due to this the attributes value pairs cannot be excluded. The biggest problem arises when a value of one of the attributes changes which does not change the page contents. Such values can be a cookie string or a colouring scheme value which is commonly encoded inside of these attribute value pairs. A minor change to one of these values would lead to a different URL, hence no match with the bookmark anymore. Unfortunately a solution to such a problem does not exist because the nature of the attribute cannot be known.

6.6 Summary

This chapter started off with a series of tests which ensured that HyperBK worked as expected according to these tests. The second part was an evaluation of the HyperBK which consists of an evaluation of the classification algorithm and the See Also results. This section then explained the problems that were found in the evaluation namely: pages with no text, different categories for the same topic, how to improve the classification and problems with http URLs.

Chapter 7: Conclusion and Future Work

HyperBK is an alternative bookmark manager for Mozilla Firefox. Unlike the standard bookmark features present in other browsers HyperBK tries to make sense of the categories and the bookmarks. It adopts a number of features to make it different from the standard bookmark managers present in browsers. First of all each bookmark can reside only once inside of the bookmark file. Each bookmark, apart from its name and URL is represented by the original page title, and a page thumbnail. Together with this HyperBK keeps a copy of the bookmarked page in the local cache in case the web page is offline. In the case the bookmarked page is offline a search query is sent to Google to try to relocate the lost bookmark file.

HyperBK offers a number of features to help the user in having an organized bookmark file. One feature is the bookmark verification utility which verifies the bookmarks and checks the status of each. Another tool (Divide Category Wizard) pops up when a bookmark category grows too large, to assist the user in dividing the category. Another feature is the 'See Also' function which generates an automatic query to locate pages similar to a category. The search query (SE Referrer) that the user used to locate a page that is subsequently bookmarked is also stored with the bookmark.

The idea of HyperBK is not to force the user to have an organized bookmark file but to help him in doing so. HyperBK will not create categories automatically, it is still the user's responsibility to create the categories and start dividing bookmarks into these categories. HyperBK will only offer a suggestion to the most appropriate category and suggest the last category bookmarked into if fails.

Regarding the History HyperBK divides it into topics like the bookmark categories. It adopts the scenario as if all pages visited would be bookmarked. This divides the pages the user visits into topics unlike the standard method adopted by browsers which use time and domain name to divide the pages.

7.1 Results Achieved

In chapter 6 an evaluation of the classification and see also algorithms was conducted. The classification results obtained are quite promising and in many cases the algorithm made a good match (67%). In fact one can say that the keywords that are extracted and represent a particular category are accountable for these results. One can add that when the matching fails in most cases the second candidate to be suggested would be the match. This gives an indication that the method suggested in section 6.4.2 would be quite useful as most probably the user will find the matching one out of the bulleted list.

An interesting result is the near hits i.e. when the suggested category is the parent of the equivalent match in the user's bookmark file. Now in HyperBK's implementation each category is treated independently of other categories that are present inside of it. This leads to the conclusion that there are cases where the content of the parent and children is similar, and there needs to be ways of detecting how a category will be allocated to a child category instead of its parent.

The See Also utility proved to be a good method of suggesting additional pages that belong to a particular category. In fact the results obtained are satisfying as 41% of the suggested results were rated as 'Good' and another 41% were rated as 'Really Good'. As only one search engine was used there is no means of checking if using another search engine might have produced better results, similarly as Google Directory was used then sites not listed in the directory are excluded from the results (Blogs, News pages etc).

7.2 Aims Achieved

In section 1.2 seven aims were put forward which were required to be reached in order to achieve project success. The aims and their results were the following:

- Providing a simple way of classifying a web document into a bookmark category:
Documents were classified into their appropriate category successfully with 67% hit according to the evaluation carried out on sample bookmark files.

This evaluation was carried out on manually organised bookmarks with no proof that they were organised in the best way. Using HyperBK on an empty bookmark file might lead to totally different results.

- Maintaining a healthy bookmark file in an organized fashion:

The bookmark file is kept organised in various ways: Each time a bookmark category exceeds its maximum limit a wizard is opened to help the user split the category into two sub categories. Similarly the verify bookmark utility will ensure that no dead bookmarks exist inside of the bookmark file.

- Easier way of locating pages in history:

Dividing the history into bookmark categories makes the history more organised as this matches more the user's mental representation [11].

- Better representation of bookmarks:

Each bookmark is represented by a page thumbnail which helps the user to recognise the bookmark better [28].

- Portable solution:

Using Mozilla Firefox all functionality can be used on three different platforms i.e. Windows, MacOS and Linux.

- Tightly coupled with browser:

Being a Firefox Extension all functionality is tightly coupled with the browser and available from within it.

- See Also Recommendations:

See Also recommendations proved to be useful as the users in the evaluation rated the results: 41% of them were rated as 'Good' and another 41% as 'Really Good'.

7.3 Future Work

This section suggests some improvements and new features that can be introduced in later versions of HyperBK.

7.3.1 See Also

A feature in HyperBK that could be enhanced is the See Also option which gives a list of similar pages for a particular category. In fact there is a vast amount of work that can be done on this part of HyperBK to provide better results to the user. The enhancement can take place on three different levels:

- Search Query Generated
- Search Engines Used
- Filtration of Results

There are various methods how the search query can be generated. In HyperBK two methods are used which are:

- from the automatically collected keywords
- from keywords of past user-defined search queries

In each method the search query is generated by collecting the top four used keywords in either case. This can be improved in lots of ways; one is considering that the most popular terms are not always the best to identify a particular category. One must also note that the method adopted does not exclude the fact that terms with the same meaning could be added to the same query, and ideally such terms would be included with the '|' (or) symbol. This would involve adding some semantic analyzer to analyze the semantic meaning of the words in a category

HyperBK utilises only one search engine i.e. Google, and in fact does not use the power of other search engines. There are lots of ways in which other search engines can be used to retrieve better results. A method described in [29] is that of using multiple search engines, getting the results and grouping them into one. Then the top results can be supplied to the user and this will increase recall. On the other hand to achieve higher precision what is done in [29] is that selected documents are extracted and a profile is generated to match the user's preferences. In HyperBK this profile already exists if one looks at the bookmark categories. By looking at the bookmark categories and at the pages they represent (available in the local cache) one can generate a detailed profile on the user. Figure 40 is a diagram adapted from

[29] to demonstrate this approach and how it can be integrated into HyperBK to offer higher precision and recall in the See Also for a particular category.

Another improvement is that of filtering the results obtained to match more what the user requires. One method is that of removing sites which have already been visited by taking a look at the history. Similarly using the history, sites which are similar to the ones visited can be given a higher score as they would in many cases be more of interest to the user.

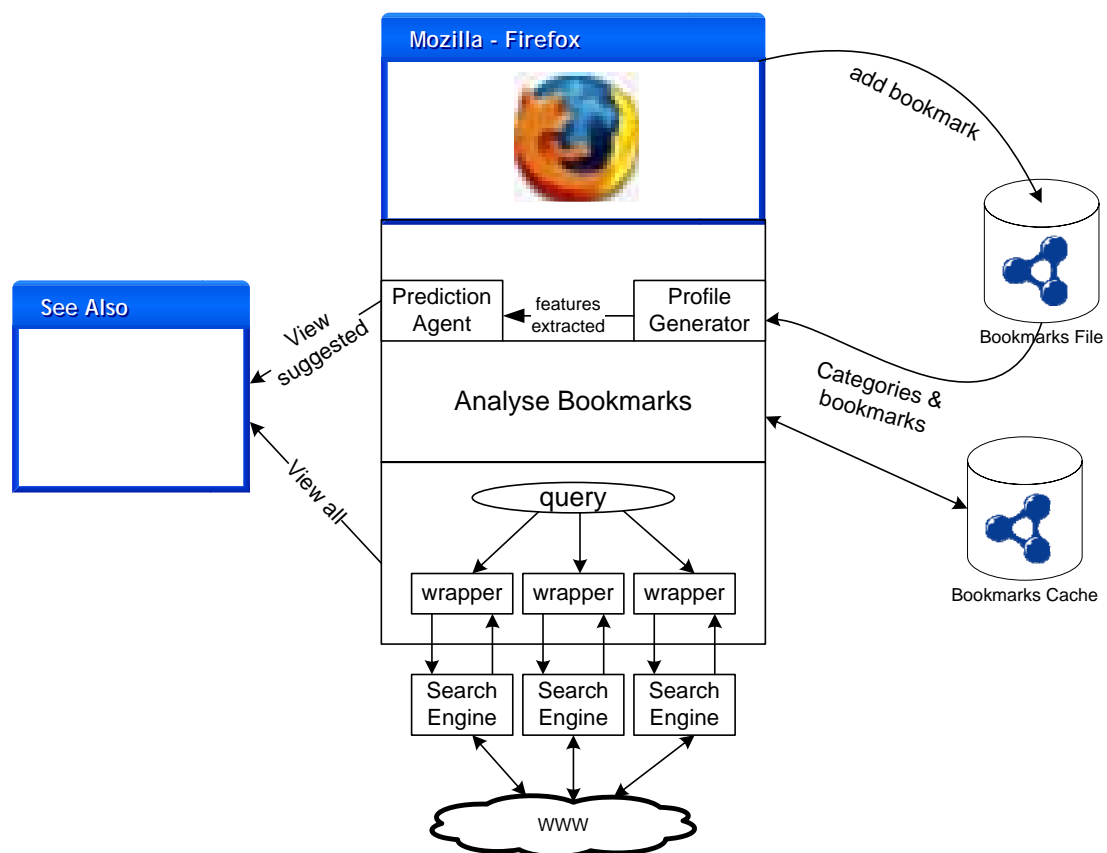


Figure 40 Suggested See Also Method (adapted from [29])

7.3.2 Detecting Page Changes

The only way HyperBK detects that a bookmarked page has been updated is by inspecting the header of the http response and checking the Last-Modified header field. Unfortunately such a header field is missing in most dynamic pages. The only

way to detect when a page has been updated is to compare the page contents with the one on disk (in the local cache). There are various ways an html page can change; some tags can change, an advertisement can be changed, or the actual page contents can change. These all have to be considered. Once page update detection can be performed then the following features can be introduced:

- updating the bookmark cache only when the page has been updated
- option to highlight updated content
- determine contents that change frequently in a page

Pages that are updated frequently in many cases contain news on some event or else they would be blogs. This can give further insight on the content that really makes up the page.

7.3.3 Support for PDF and DOC

A lot of online documents are available in pdf or word doc format. Currently HyperBK does not parse any of the content of such documents. Being capable of parsing such documents would enable classification of these documents into the appropriate category.

7.3.4 Higher UI Functionality

Drag and drop are useful features for power users when it comes to moving bookmarks from one folder to another. Similarly for deletion of bookmarks a bookmark can be moved to a 'Trash' folder. Bookmarking can be extended to dragging a link onto the Add Bookmark button, or directly into the HyperBK sidebar onto the appropriate category.

7.3.5 Power of the Semantic Web

With the use of ontologies, classification of a web page may become much easier than it is now as there would be no need to collect the top five terms of a page (see section 5.3) but instead the ontologies can be used. The problem with this is that it will take some time before web page authors will start including such semantic related stuff in the www documents. This approach can be used in conjunction with

the page keywords so that documents which have no binded ontologies can be categorised just the same to their matching category.

7.3.6 Relation between bookmarked pages and history pages

In the current history view (ref. section 5.8.1) all pages are shown by default in the best matched category. The filters available can filter out by date, keyword or period but an extra filter on content might be useful. Such a filter would show only those pages which have a direct link to the bookmark pages. Direct link indicates that the page is a few nodes away from some bookmarked page.

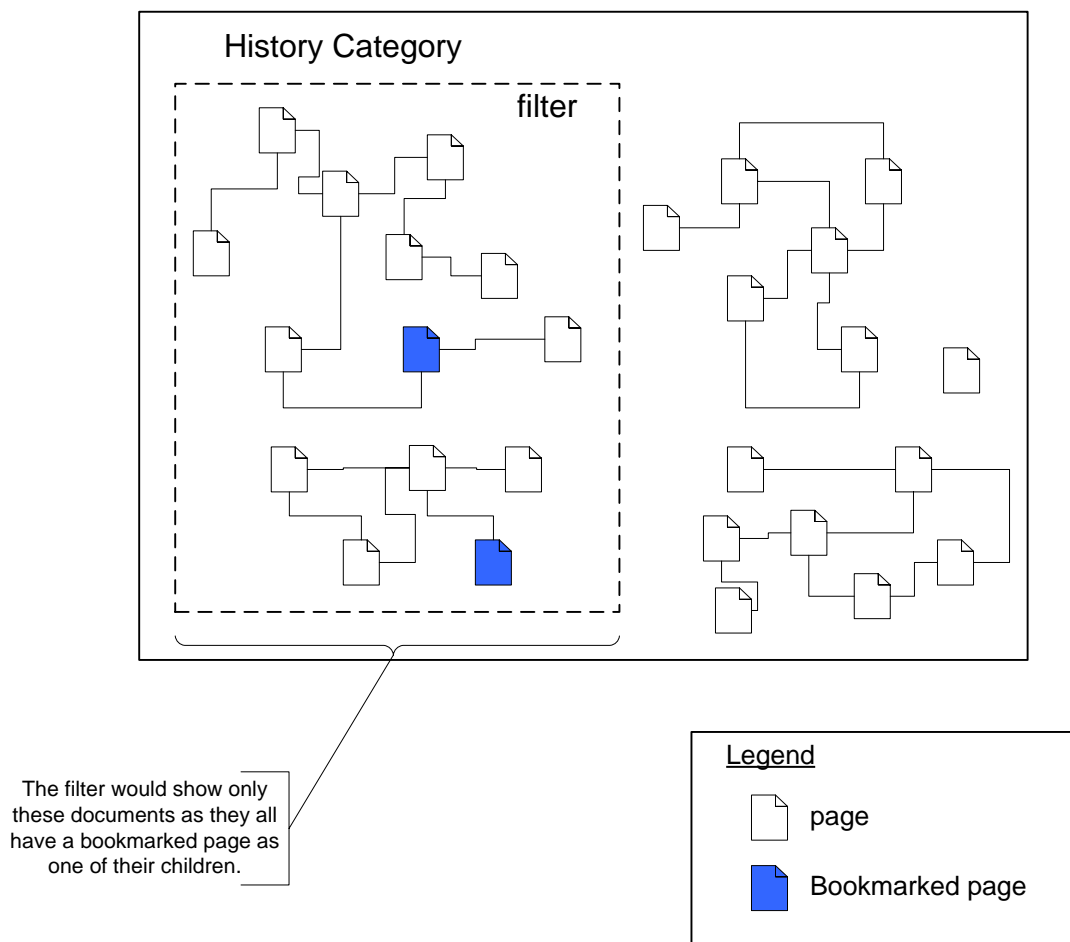


Figure 41 History with bookmarked pages & their relations

Figure 41 depicts this approach, where the elements inside the dotted box will be the history list for that particular category when this particular filter is applied. This filter

would exclude the other documents, which although similar and on the same topic do not link to any bookmarked page.

The amount of computation required for such a feature is quite expensive as all pages inside of a history category would have to be parsed and checked, but there is another approach which can give good results and requires fewer resources. This involves using features such as the link: and related: keywords in a search engine query. This approach would require performing a query for each document and getting in return a list of URLs that link to that page. Using this list the graph of the documents in hyperspace and how they link to each other can be constructed.

Similarly using this approach this filter can be applied on any page not only bookmarked one. This will make it possible for a user to locate a page in history if s/he knows that some page lead to another but possible the visit times were different, thus making it not possible to get the page using the filter by same time period.

7.3.7 Bookmark/History Sharing and Online Access

As users normally use different machines to browse the WWW it would be useful if they could access their bookmark files from anywhere. In fact this feature has started to gain popularity and many providers are offering such a service for free (Yahoo, Google, Microsoft etc). The main idea would be that of having the bookmark file being accessible remotely and at application startup this would be downloaded, and re-uploaded on application exit. As the bookmark and history file are in RDF this is ready to be ported from one machine to another.

Having the bookmark available at a centralised location can lead to other possible features being offered. A simple feature would be to share the bookmarks through anonymous means. Secondly when a user requests a 'See Also' query for a particular category the system can look at similar categories from other users and extract some of the documents obtained. Obviously there is a great deal to keep user's privacy at stake and each bookmark file would have to be anonymous. But on

the other hand this approach can pick up pages which are categorised differently to automatic categorization.

References

- [1] M. Andreessen, E. Bina, 'Mosaic Web Browser History – NCSA',
http://www.livinginternet.com/w/wi_mosaic.htm [28th April 2006]

- [2] 'IE 7.0 - A history of browsers',
<http://www.quirksmode.org/browsers/history.html> [28th April 2006]

- [3] 'Firefox 1.0 released | TG Daily'
http://www.tgdaily.com/2004/11/09/firefox_1/index.html [28th April 2006]

- [4] Robert Hobbes' Zakon, Zakon Group LLC, 'Hobbes' Internet Timeline - the definitive ARPAnet & Internet history',
<http://www.zakon.org/robert/internet/timeline/> [28th April 2006]

- [5] D. Abrams, R. Baecker, M. Chignell, (1998) 'Information Archiving with Bookmarks: Personal Web Space Construction and Organization'. *In Proceeding of CHI'98*

- [6] W. Jones, H. Bruce, S. Dumais, (October 2001) 'Keeping Found Things Found on the Web'. *In Proceedings of ACM's CTKM'01, Tenth International Conference on Information and Knowledge Management*, 119-126

- [7] D. Abrams & R. Baecker, (1997) 'How people use WWW Bookmarks'. *ACM SIGCHI 1997 Conference*

- [8] L. Tauscher, S. Greenberg, (March 1997) 'Revisitation Patterns in World Wide Web Navigation'. *In Proceedings of the Conference on Human Factors in Computing Systems CHI'97*.

- [9] S. Kaasten, S. Greenberg, (March 2001) 'Integrating Back, History and Bookmarks in Web Browsers'. *Conference on Human Factors in Computing Systems*.
- [10] S. LeeTiernan, S. Farnham, L. Cheng, (April 2003) 'Two Methods for Auto-Organizing Personal Web History'. *CHI '03 extended abstracts on Human factors in computing systems*.
- [11] J. Gemmell, G. Bell, R. Lueder, S. Drucker, C. Wong, (December 2002) 'MyLifeBits: Fulfilling the Memex Vision'. *In Proceedings of the tenth ACM international conference on Multimedia*
- [12] BookmarkTracker, <http://www.bookmarktracker.com> [28th April 2006]
- [13] ikeepbookmarks.com, <http://www.ikeepbookmarks.com> [28th April 2006]
- [14] P. Kokosis, V. Krikos, S. Stamou, D. Christodoulakis, (June 2005) 'HiBO: A System for Automatically Organizing Bookmarks'. *In Proceedings of the 5th ACM/IEEE-CS joint conference on Digital libraries*
- [15] Check&Get - Bookmark Manager, <http://activeurls.com/en/> [28th April 2006]
- [16] M. Hascoet, 'Navigation and interaction within graphical bookmarks'. University of Paris-sud, France. <http://citeseer.ist.psu.edu/296052.html>
- [17] H. Li, K. Yamanishi, (July 1997) 'Document Classification using a Finite Mixture Model'. *In Proceedings of the 35th annual meeting on Association for Computational Linguistics*.
- [18] D. Shen, Z. Chen, Q. Yang, H. Zeng, B. Zhang, Y. Lu, W. Ma, (July 2004) 'Web-page Classification though Summarization' *In Proceedings of the 27th*

annual international ACM SIGIR conference on Research and development in information retrieval.

- [19] Martin Porter, 'Porter Stemming Algorithm'
<http://www.tartarus.org/martin/PorterStemmer/> [28th April 2006]
- [20] J. Novovicova , 'Text Document Classification',
http://www.ercim.org/publication/Ercim_News/enw62/novovicova.html [28th April 2006]
- [21] G. Salton, (1968) 'Automatic Information Organization and Retrieval'. McGraw-Hill, New York, 18.
- [22] C.J. van Rijsbergen, (1979) 'Information Retrieval' – 'Chapter 3 Automatic Classification', <http://www.dcs.gla.ac.uk/Keith/Chapter.3/Ch.3.html> [28th April 2006]
- [23] 'Naïve Bayes Classifier'
http://www.absoluteastronomy.com/reference/naive_bayes_classifier [28th April 2006]
- [24] 'Latent Semantic Analysis'
http://www.absoluteastronomy.com/reference/latent_semantic_analysis [28th April 2006]
- [25] Melodie Grima (February 2003) 'BMAC – Bookmark Management system with Automated Control, Literature Review' *Unpublished*
- [26] 'JavaScript - Wikipedia, the free encyclopedia' <http://en.wikipedia.org/wiki/Js> [28th April 2006]

- [27] 'thecounter.com The Full-Featured Web Counter with Graphic Reports and Detailed Information', <http://www.thecounter.com/stats/2006/March/browser.php>
[28th April 2006]
- [28] S. Kaasten, S. Greenberg, C. Edwards, (2001) 'How People Recognize Previously Seen Web Pages from Titles, URLs and Thumbnails'. Report 2001-692-15; Dept. of Computer Science, Univ. of Calgary, Alberta, Canada, 2001.
- [29] M. Montebello, (August 1998) 'Optimizing Recall/Precision scores in IR over the WWW'. *In Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval.*

Bibliography

'XULPlanet.com' - <http://www.xulplanet.com>

'Mozilla Developer Center – MDC' - <http://developer.mozilla.org>

Nigel McFarlane, (2003) '*Rapid Application Development with Mozilla*', Prentice Hall PTR

Nigel McFarlane, (2005) '*Firefox Hacks*', O'Reilly

Peter D. Hipson, (2005) '*Firefox and Thunderbird: Beyond Browsing and Email*', Que

Glossary

ASCII	American Standard Code for Information Interchange
COM	Component Object Model
CSS	Cascading Style Sheets
DOM	Document Object Model
HTML	Hypertext Markup Language
IR	Information Retrieval
RDF	Resource Description Framework
RSS	Rich Site Summary
SOAP	Simple Object Access Protocol
TF	Term Frequency
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WWW	World Wide Web
XML	Extensible Markup Language
XPCOM	Cross Platform Component Object Model
XUL	XML User Interface

Appendix A: Further Implementation Details

RDF Datasources

Bookmarks

Name	Property
Bookmark Name	http://home.netscape.com/NC-rdf#Name
Page URL	http://home.netscape.com/NC-rdf#URL
Page Title	http://home.netscape.com/NC-rdf#Page
Bookmark Visit Count	http://home.netscape.com/NC-rdf#VisitCount
Page Favicon	http://home.netscape.com/NC-rdf#Icon
Bookmark Add Date	http://home.netscape.com/NC-rdf#BookmarkAddDate
Bookmark Last Visit Date	http://home.netscape.com/WEB-rdf#LastVisitDate
Page Keywords (stemmed)	http://home.netscape.com/NC-rdf#Keywords
Page Keywords	http://home.netscape.com/NC-rdf#OrigKeywords
Page Referrer	http://home.netscape.com/NC-rdf#Referrer
URL in cache	http://home.netscape.com/NC-rdf#ShortcutURL
Thumbnail URL	http://home.netscape.com/NC-rdf#Image

Fast Bookmarks

Name	Property
Fast Bookmark Name	http://home.netscape.com/NC-rdf#Name
Page URL	http://home.netscape.com/NC-rdf#URL
Page Title	http://home.netscape.com/NC-rdf#Page
Fast Bookmark Add Date	http://home.netscape.com/NC-rdf#BookmarkAddDate
Page Favicon	http://home.netscape.com/NC-rdf#Icon

History

Name	Property
Page Title	http://home.netscape.com/NC-rdf#Name
Page URL	http://home.netscape.com/NC-rdf#URL
Page Last Visit Date	http://home.netscape.com/WEB-rdf#LastVisitDate
Page Keywords (stemmed)	http://home.netscape.com/NC-rdf#Keywords
Page Keywords	http://home.netscape.com/NC-rdf#OrigKeywords
Page Referrer	http://home.netscape.com/NC-rdf#Referrer
Page Visit Counts	http://home.netscape.com/NC-rdf#VisitCount

Common
English Words

1 the	50 she	102 new	154 want
2 of	51 which	103 work	155 air
3 to	52 do	104 part	156 well
4 and	53 their	105 take	157 also
5 null	54 time	106 get	158 play
6 in	55 if	107 place	159 small
7 is	56 will	108 made	160 end
8 it	57 way	109 live	161 put
9 you	58 about	110 where	162 home
10 that	59 many	111 after	163 read
11 he	60 then	112 back	164 hand
12 was	61 them	113 little	165 port
13 for	62 would	114 only	166 large
14 on	63 write	115 round	167 spell
15 are	64 like	116 man	168 add
16 with	65 so	117 year	169 even
17 as	66 these	118 came	170 land
18 com	67 her	119 show	171 here
19 his	68 long	120 every	172 must
20 they	69 make	121 good	173 big
21 be	70 thing	122 me	174 high
22 at	71 see	123 give	175 such
23 one	72 him	124 our	176 follow
24 have	73 two	125 under	177 act
25 this	74 has	126 name	178 why
26 from	75 look	127 very	179 ask
27 or	76 more	128 through	180 men
28 had	77 day	129 just	181 change
29 by	78 could	130 form	182 went
30 hot	79 go	131 much	183 light
31 but	80 come	132 great	184 kind
32 some	81 did	133 think	185 off
33 what	82 my	134 say	186 need
34 there	83 sound	135 help	187 house
35 we	84 no	136 low	188 picture
36 can	85 most	137 line	189 try
37 out	86 number	138 before	190 us
38 other	87 who	139 turn	191 again
39 were	88 over	140 cause	192 animal
40 all	89 know	141 same	193 point
41 your	90 water	142 mean	194 mother
42 when	91 than	143 differ	195 world
43 up	92 call	144 move	196 near
44 use	93 first	145 right	197 build
45 word	94 people	146 boy	198 self
46 how	95 may	147 old	199 earth
47 said	96 down	148 too	200 father
48 an	97 side	149 does	201 head
49 each	98 been	150 tell	202 stand
	99 now	151 sentence	203 own
	100 find	152 set	204 page
	101 any	153 three	205 should

206	country	258	got	310	list	362	hold
207	found	259	walk	311	though	363	west
208	answer	260	example	312	feel	364	ground
209	school	261	ease	313	talk	365	interest
210	grow	262	paper	314	bird	366	reach
211	study	263	often	315	soon	367	fast
212	still	264	always	316	body	368	five
213	learn	265	music	317	dog	369	sing
214	plant	266	those	318	family	370	listen
215	cover	267	both	319	direct	371	six
216	food	268	mark	320	pose	372	table
217	sun	269	book	321	leave	373	travel
218	four	270	letter	322	song	374	less
219	thought	271	until	323	measure	375	morning
220	let	272	mile	324	state	376	ten
221	keep	273	river	325	product	377	simple
222	eye	274	car	326	black	378	several
223	never	275	feet	327	short	379	vowel
224	last	276	care	328	numeral	380	toward
225	door	277	second	329	class	381	war
226	between	278	group	330	wind	382	lay
227	city	279	carry	331	question	383	against
228	tree	280	took	332	happen	384	pattern
229	cross	281	rain	333	complete	385	slow
230	since	282	eat	334	ship	386	center
231	hard	283	room	335	area	387	love
232	start	284	friend	336	half	388	person
233	might	285	began	337	rock	389	money
234	story	286	idea	338	order	390	serve
235	saw	287	fish	339	fire	391	appear
236	far	288	mountain	340	south	392	road
237	sea	289	north	341	problem	393	map
238	draw	290	once	342	piece	394	science
239	left	291	base	343	told	395	rule
240	late	292	hear	344	knew	396	govern
241	run	293	horse	345	pass	397	pull
242	don't	294	cut	346	farm	398	cold
243	while	295	sure	347	top	399	notice
244	press	296	watch	348	whole	400	voice
245	close	297	color	349	king	401	fall
246	night	298	face	350	size	402	power
247	real	299	wood	351	heard	403	town
248	life	300	main	352	best	404	fine
249	few	301	enough	353	hour	405	certain
250	stop	302	plain	354	better	406	fly
251	open	303	girl	355	true .	407	unit
252	seem	304	usual	356	during	408	lead
253	together	305	young	357	hundred	409	cry
254	next	306	ready	358	am	410	dark
255	white	307	above	359	remember	411	machine
256	children	308	ever	360	step	412	note
257	begin	309	red	361	early	413	wait

414	plan	466	foot
415	figure	467	yet
416	star	468	busy
417	box	469	test
418	noun	470	record
419	field	471	boat
420	rest	472	common
421	correct	473	gold
422	able	474	possible
423	pound	475	plane
424	done	476	age
425	beauty	477	dry
426	drive	478	wonder
427	stood	479	laugh
428	contain	480	thousand
429	front	481	ago
430	teach	482	ran
431	week	483	check
432	final	484	game
433	gave	485	shape
434	green	486	yes
435	oh	487	hot
436	quick	488	miss
437	develop	489	brought
438	sleep	490	heat
439	warm	491	snow
440	free	492	bed
441	minute	493	bring
442	strong	494	sit
443	special	495	perhaps
444	mind	496	fill
445	behind	497	east
446	clear	498	weight
447	tail	499	language
448	produce	500	among
449	fact	501	click
450	street	502	www
451	inch		
452	lot		
453	nothing		
454	course		
455	stay		
456	wheel		
457	full		
458	force		
459	blue		
460	object		
461	decide		
462	surface		
463	deep		
464	moon		
465	island		

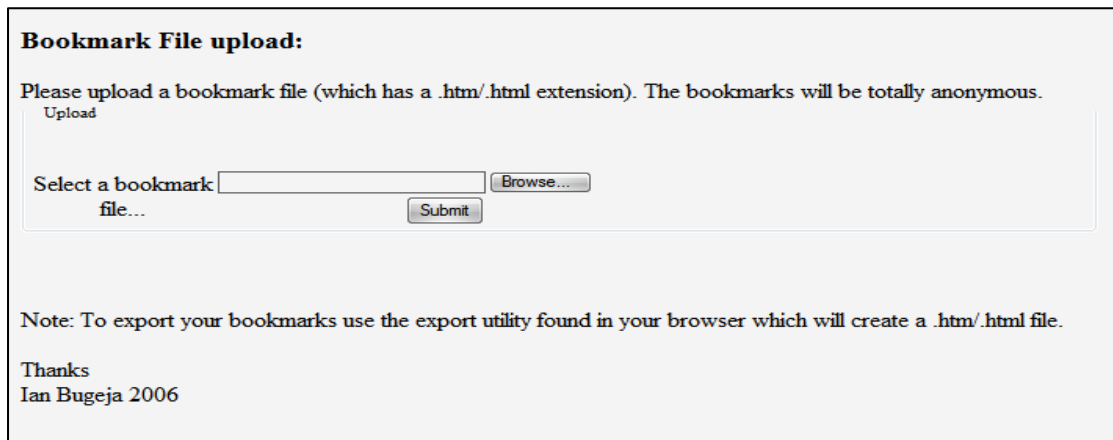
<http://www.world-english.org/english500.htm>

Appendix B: Evaluation Details

For the evaluation to take place a website was used to submit bookmark files and to get feedback from the volunteers on the 'See Also' results. This was the best way to keep the volunteers anonymous and have an easier method of submission of bookmark files.

Bookmark file submission

The bookmark submission page (Figure 42) was required so that volunteers could submit bookmark files anonymously. Html bookmark files were accepted, which are the format Microsoft IE and Firefox export to. Once the bookmark file was uploaded the system returned a unique ID to the volunteers through which they could identify themselves. This ID was used in the second part of the evaluation which involved the 'See Also' recommendations.



Bookmark File upload:

Please upload a bookmark file (which has a .htm/.html extension). The bookmarks will be totally anonymous.

Upload

Select a bookmark file...

Note: To export your bookmarks use the export utility found in your browser which will create a .htm/.html file.

Thanks
Ian Bugeja 2006

Figure 42 Bookmark File Submission Page

See Also

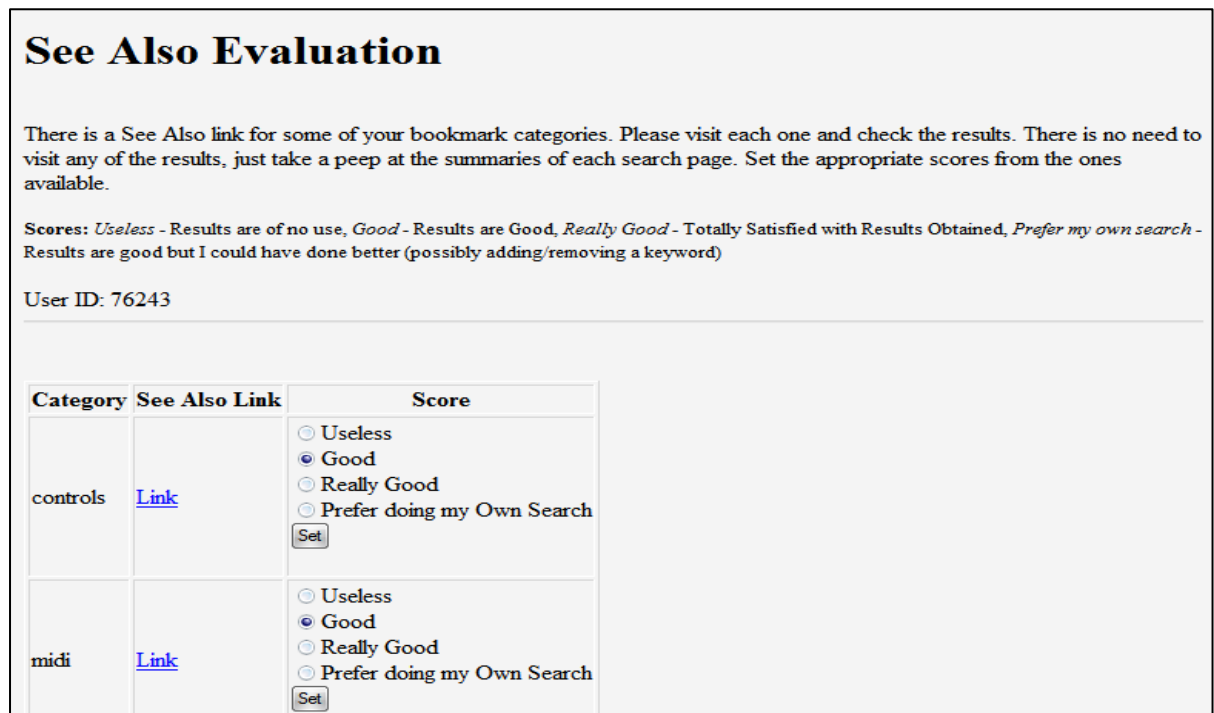
Once that bookmark file was imported into HyperBK and each bookmark was visited then a See Also query was generated for around 3-4 random categories that are present in the bookmark file.



The image shows a web page titled "See Also Evaluation". Below the title, there is a text prompt "Please Enter your ID:". Underneath this prompt is a text input field, followed by two buttons labeled "Go" and "Reset". At the bottom left of the page, it says "Ian Bugeja 2006".

Figure 43 See Also Evaluation Login Page

Each volunteer was then asked to visit the page shown in Figure 43 and here s/he was required to enter the ID which was returned when the bookmark file was submitted. On entering of the ID and clicking on the Go button the following page (Figure 44) was displayed.



The image shows a web page titled "See Also Evaluation". Below the title, there is a paragraph of instructions: "There is a See Also link for some of your bookmark categories. Please visit each one and check the results. There is no need to visit any of the results, just take a peep at the summaries of each search page. Set the appropriate scores from the ones available." Below this is a line of text: "Scores: *Useless* - Results are of no use, *Good* - Results are Good, *Really Good* - Totally Satisfied with Results Obtained, *Prefer my own search* - Results are good but I could have done better (possibly adding/removing a keyword)". Below that is "User ID: 76243". The main part of the page is a table with three columns: "Category", "See Also Link", and "Score". There are two rows of data. The first row has "controls" in the "Category" column, a blue "Link" in the "See Also Link" column, and a "Score" column with four radio buttons: "Useless", "Good" (which is selected), "Really Good", and "Prefer doing my Own Search", followed by a "Set" button. The second row has "midi" in the "Category" column, a blue "Link" in the "See Also Link" column, and a "Score" column with the same four radio buttons and a "Set" button.

Category	See Also Link	Score
controls	Link	<input type="radio"/> Useless <input checked="" type="radio"/> Good <input type="radio"/> Really Good <input type="radio"/> Prefer doing my Own Search <input type="button" value="Set"/>
midi	Link	<input type="radio"/> Useless <input checked="" type="radio"/> Good <input type="radio"/> Really Good <input type="radio"/> Prefer doing my Own Search <input type="button" value="Set"/>

Figure 44 See Also Evaluation

In this page a table is shown which consists of 3 columns. The first column is the name of the category that the equivalent See Also query was generated for. The second was the link to the query which would open a new window with a Google Directory Search. The third column is the score that is required to be adjusted according to how the search results are. The score to select was one out of the following:

- *Useless* - Results are of no use
- *Good* - Results are Good
- *Really Good* - Totally Satisfied with Results Obtained
- *Prefer my own search* - Results are good but I could have done better (possibly adding/removing a keyword)

Once clicking on the Set button the result is stored. The above procedure had to be done for each category listed. This completed the See Also Evaluation so that the results could be examined.

Development

These webpages were developed in PHP v5. The first page was a simple upload file script. The bookmark file once uploaded was renamed to the ID (5digit) generated for that particular volunteer. This ensured that there were no duplicate copies of the bookmark names and that the volunteer was kept anonymous.

The See Also data (user id, category name, category see also link and score) were stored in an Microsoft Access database and then using sql queried and displayed in the html according to the ID given.

All of these files can be found on the CD attached with this document in the folder 'evaluation'.

Appendix C: HyperBK - User Manual

Welcome to HyperBK, a new and innovative bookmark and history manager for Mozilla Firefox. The HyperBK extension is designed to be used by anybody as a better way of maintaining the bookmarks and an easier way of locating past visited pages.

1.1 Installation

Installation is very easy; the only prerequisite is that you have Firefox 1.5 installed on your system be it Windows, Linux or MacOS. To install the extension simply visit <http://hyper.iannet.org> and click on 'Install Now'. This will popup a dialog like Figure 45 and after a delay of four seconds the Install Now button will become active making it possible click it and install HyperBK.

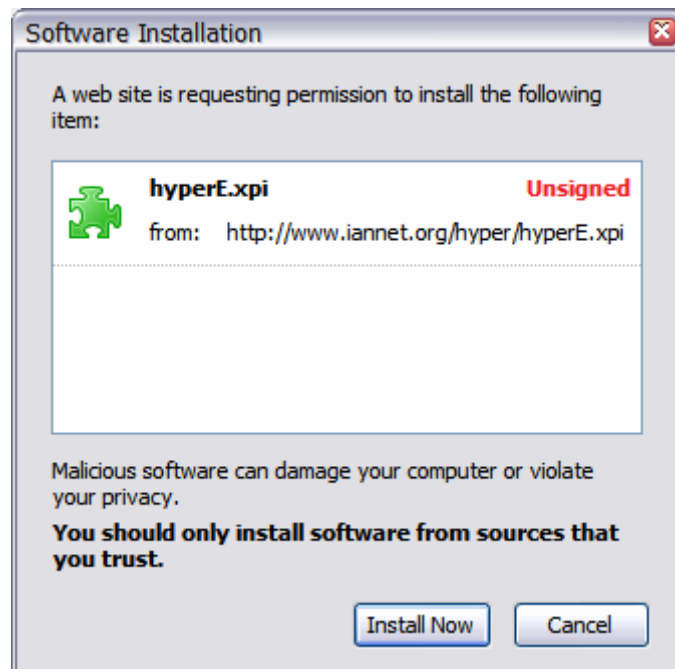


Figure 45 Install Caption Dialog

Firefox will need to be restarted in order to complete the installation.

1.2 Welcome Wizard

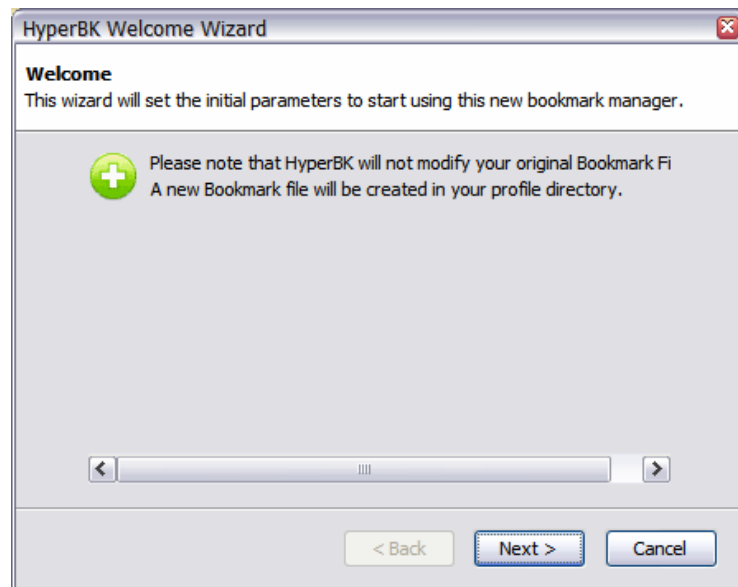


Figure 46 Welcome Wizard Page 1

Once HyperBK has been installed, the first time Firefox is run, a wizard (Figure 46) will pop up. This wizard will be used to set some initial parameters for HyperBK. Pressing on next will open the next page (Figure 47).

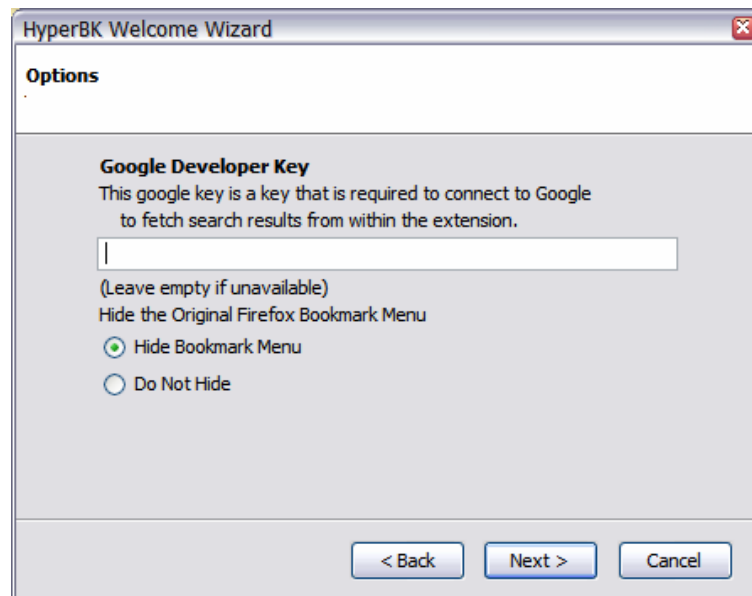


Figure 47 Welcome Wizard Page 2

In this wizard page enter a Google developer key. This can be obtained from www.google.com/apis. This key is used in the see also and bookmark relocation. If no Google developer key is available then just leave this entry empty. It can also be entered from HyperBK preferences later on. The second option is required if you want to still view the standard Firefox bookmark menu.

Please note that in any way HyperBK will tamper with the original Firefox bookmarks. All new bookmarks will be stored in separate files, and if the necessity arises uninstalling HyperBK will regain normal Firefox operation.

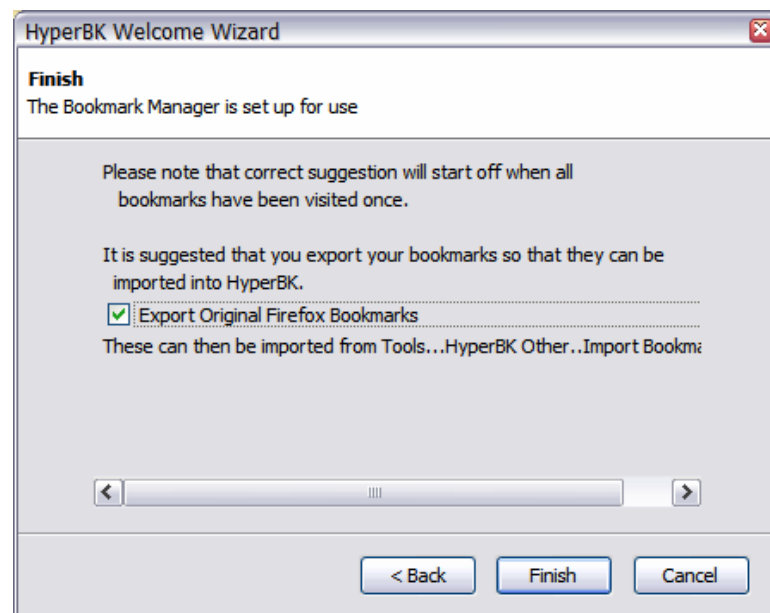


Figure 48 Welcome Wizard Page 3

Figure 48 shows the final wizard page. The option to export the Firefox bookmarks is useful in case you want to import them into HyperBK. Unselect it if you do not want to export the Firefox bookmarks.

1.3 Import Bookmarks

To import bookmarks from an html file do the following:

1. Go to Tools...HyperBK Other...Import Bookmarks

2. Select the file you want to import. This file will be shown in the window
3. Select Yes to import the bookmarks

For HyperBK to use its full capabilities then each bookmark must be visited at least once. This can be done using the First Bookmark Tour available from Tools...HyperBK Other...First Bookmark Visit Tour.

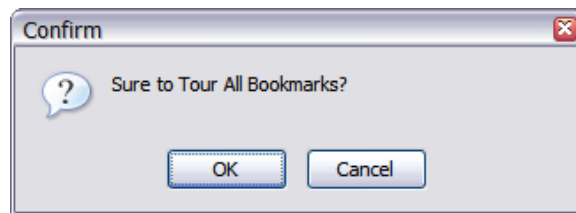


Figure 49 Tour All Bookmarks Prompt

When the tour is selected a dialog (Figure 49) will be popped up. Click on OK to tour all bookmarks that have not yet been visited. Image loading is disabled throughout the tour to make the webpages load faster.

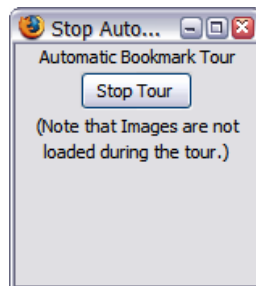


Figure 50 Stop Tour Window

The tour can be stopped by clicking on the 'Stop Tour' button in the window shown in Figure 50. Clicking to stop the tour will automatically reset the image loading so that images will now load once again. You can easily stop and resume a tour, as in this tour only bookmarks that have not yet been visited will be actually visited.

1.4 Adding Bookmarks

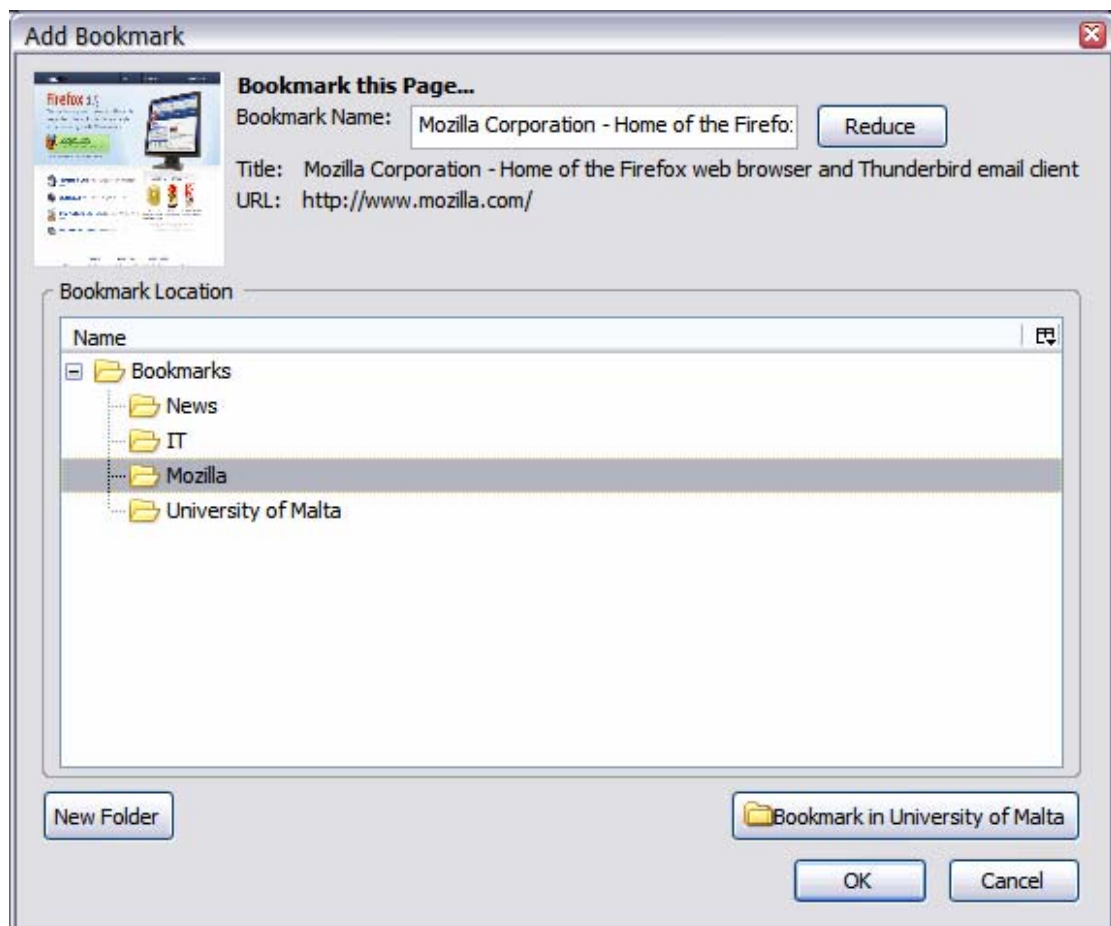


Figure 51 Add Bookmark Dialog

To add a bookmark press Ctrl-D, click the 'Bookmark This Page...', or click on 'Add Bookmark' on HyperBK toolbar. On clicking any of these commands the dialog shown in Figure 51 is displayed.

This dialog consists of the following:

- Page Thumbnail
- Bookmark Name
- Bookmark Name Reduce Button
- Bookmark Page title
- Bookmark Page URL
- Category Tree

- New Folder Button
- Bookmark in Last Bookmarked Category Button
- OK and Cancel Buttons

Using the classification algorithm the most ideal matching category will be selected. In case that the classification yields to no result then the last category a bookmark was added to will be selected. There is a button which will be used to bookmark in the last category bookmarked in. Clicking on this button will automatically bookmark in the category displayed in the button name.

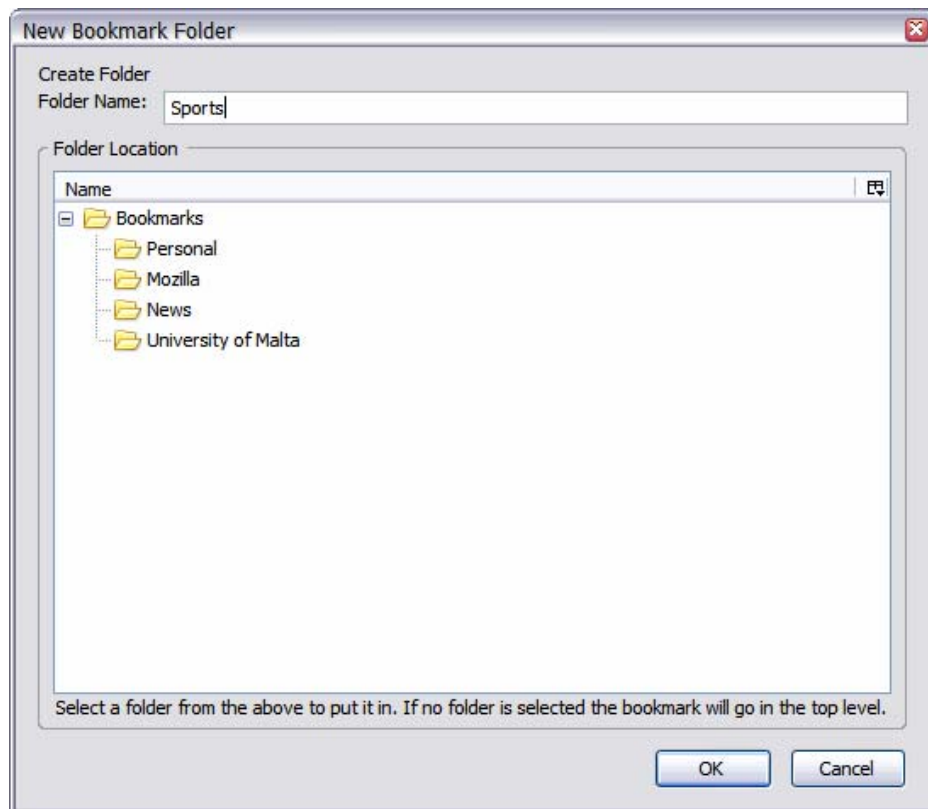


Figure 52 New Bookmark Folder Dialog

The New folder button is used to create a new bookmark category. Clicking on it will open up the New Bookmark Folder Dialog (Figure 52) which can be used to create a new category. Simply enter the desired name and select the parent folder from the folder tree. Clicking on OK will create the folder.

1.5 Bookmarks and Fast Bookmarks

Bookmarks are forever, fast bookmarks are not. This is the main difference between the two. If a page needs to be bookmarked for the next 2 or 3 sessions just add it in the fast bookmarks, no need to fill the bookmark file with useless bookmarks.

The fast bookmark is a linear list of bookmarks, no categories, no thumbnails, and the name of a page cannot be changed. The only thing to take care of is that the list can only contain a limited number of bookmarks and if this limit is exceeded the first page in will be removed. The number of bookmarks that can reside in the fast bookmarks can be adjusted from the HyperBK Preferences (see section 1.12).

1.5.1 Top Bookmarks List

This is a list of the top 10 accessed bookmarks. This list is located in the HyperBK bar.

1.6 HyperBK Browser Components

1.6.1 HyperBK Bookmark Menu

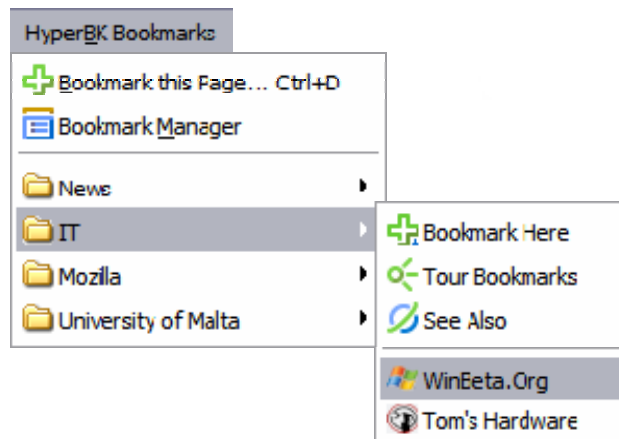


Figure 53 HyperBK Bookmark Menu

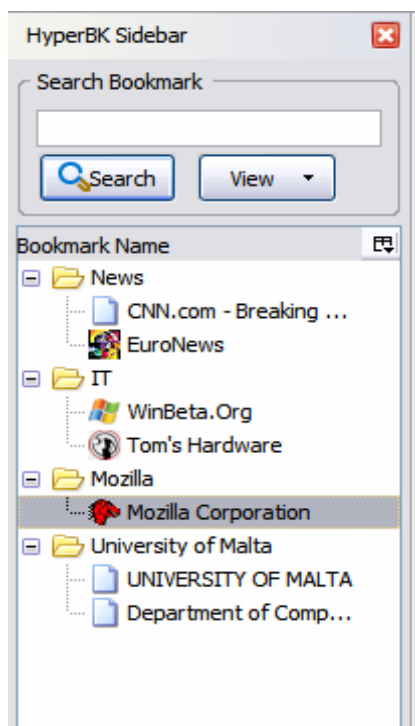
This menu (Figure 53) contains all the bookmarks that are available in HyperBK. Each submenu is a unique category which has these three commands at the top:

- **Bookmark Here**
Add the currently loaded page to this category. (No dialog is displayed)
- **Tour Bookmarks**
Start a tour of the bookmarks that reside in that category. The tour can be moved to the next/previous webpage from the ← and → on HyperBK toolbar.
- **See Also**
Perform a search query to Google to retrieve a set of similar pages to the current ones in this category. (See section 1.13)

1.6.2 HyperBK Sidebar

The browser sidebar (Figure 54) displays the bookmarks and history and has search features to search from the bookmark or history. Views can be changed by clicking on the View button and either selecting bookmarks or history. Double clicking on any entry of the tree will open the bookmark in the browser window. This sidebar can be displayed using the Shift-F2 key sequence.

For bookmarks it is possible to right click and select one of the following commands from the context menu.



▫ *Open*

Open the bookmark in the current window

▫ *Start tour from here*

Starts a bookmark tour, the tour can be moved to the next previous from the ← and → on the HyperBK toolbar.

▫ *Sort*

Sorts the entries in that bookmark folder

▫ *Rename*

Renames the bookmark/category

▫ *Delete*

Deletes the bookmark/category

Figure 54 HyperBK Browser Sidebar

1.6.3 HyperBK Toolbar

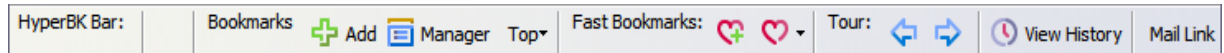


Figure 55 HyperBK Browser Toolbar

The HyperBK toolbar (Figure 55) can be displayed using the Shift-F1 key sequence or else by View...Toolbars...HyperBK Toolbar.

The toolbar contains the following elements:

- Bookmarks
 - Add – pops out Add Bookmark Dialog
 - Manager – opens Bookmark Manager with page thumbnails
 - Top – menu with top visited bookmarks
- Fast Bookmarks
 - Add – add fast bookmark
 - Menu – menu with fast bookmarks
- Tour
 - Previous
 - Next
- View History – opens History Viewer
- Mail Link – emails current open page (as link)

1.7 HyperBK Bookmark Manager

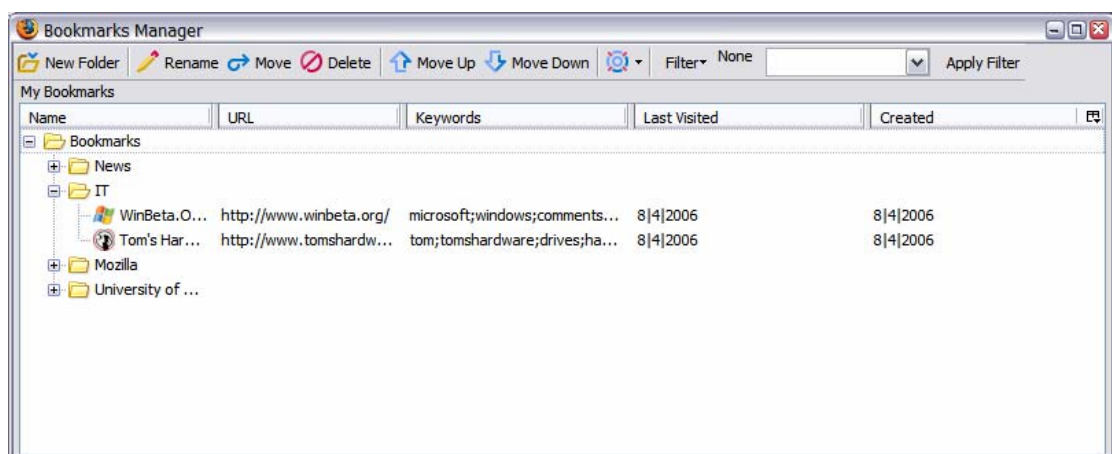


Figure 56 HyperBK Bookmark Manager Window

This bookmark manager (Figure 56) contains all the required features that a simple bookmark manager has, which include:

- Create New Folder
- Rename
- Move
- Delete
- Move Up
- Move Down
- Other
 - Sort
 - Export
- Filter

Available Filters

- By Visited Times
Enter the score of the number of times a bookmark has been visited. This is useful if the top bookmarks need to be viewed.
- By Keyword
Perform a search for a bookmark that matches the keyword. This keyword can match anything from page title, page URL, and page keywords.
- By Date
Enter the date to view pages that were visited on a specific date.

1.8 HyperBK Bookmark Manager with Page Thumbnails

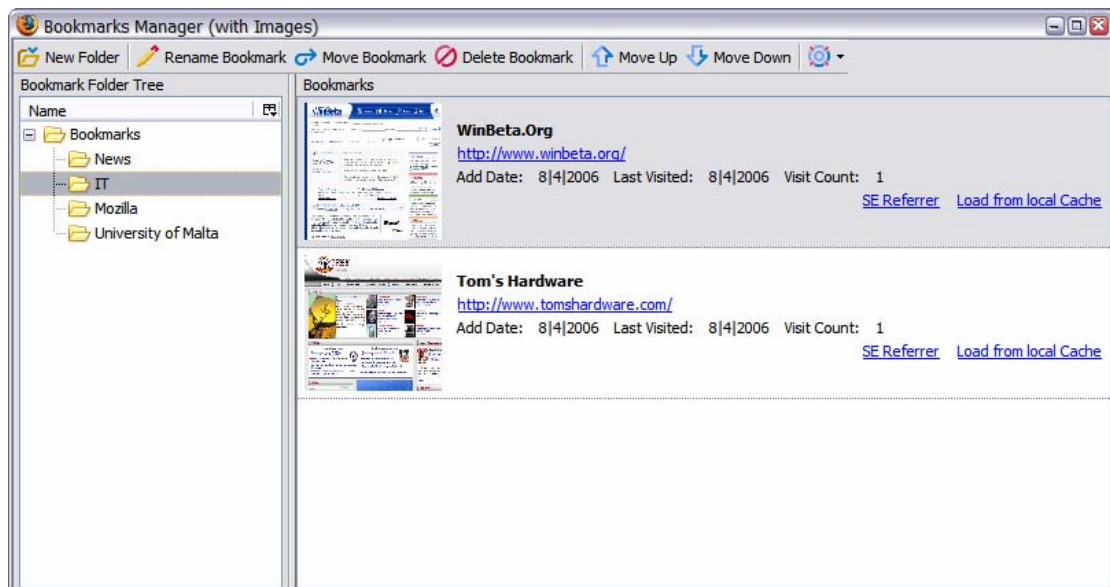


Figure 57 HyperBK Bookmark Manager (with page thumbnails) Window

Figure 57 shows another bookmark manager. It is much more user-friendly and contains the features as the previous one. It takes a different approach to the bookmarks as this time there is a page shot of the bookmark. The features offered are like the previous bookmark manager, which include Delete, Rename, and Move. Other options include:

- *SE Referrer*: This link refers to the search query used to find the page, in case the page was bookmark after a search engine query.
- *Load from local Cache*: This would load a copy of the web page which resides in the local cache. Useful option when the bookmark is offline.

1.9 Verify Bookmarks

This is a tool to verify which bookmarks are still active, which have been updated and those that are offline. To open this utility goes to Tools...HyperBK Verify. This will open a window like the one shown in Figure 58.

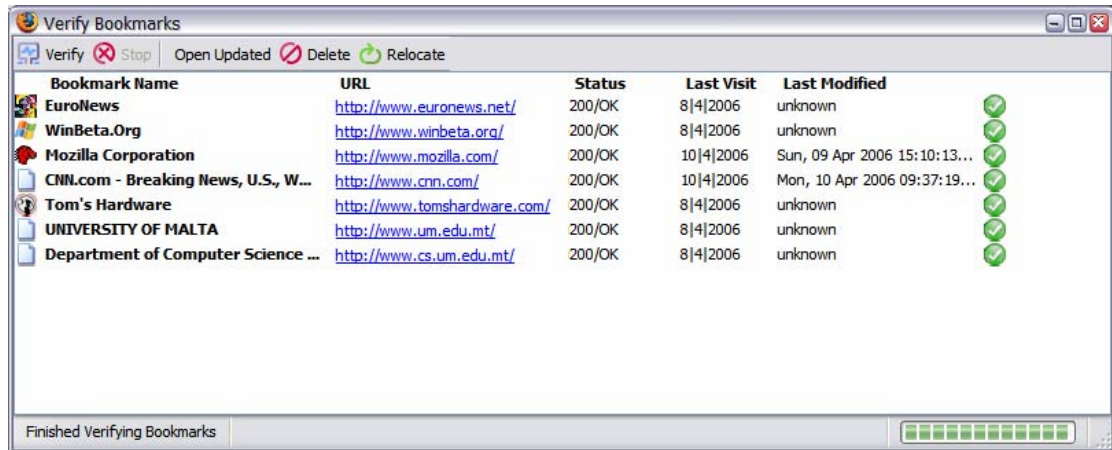


Figure 58 HyperBK Bookmark Verify Utility Window

Pressing on the Verify button will initiate a check of all bookmarks to and see the result of each bookmark. The operation can be stopped at any time using the Stop button. The other functions that are available are

- Open Updated which will open all bookmarks that were updated
- Delete which will delete the selected bookmark
- Relocate which will try to relocate the bookmark, by constructing a search query to Google.

1.10 Divide Category Wizard

Whenever the bookmarks found inside a category exceed the allowable limit, then the Divide Category Wizard (Figure 59) will be popped open. This will divide a particular category into two. You can set the limit that a category can hold in HyperBK Preferences.

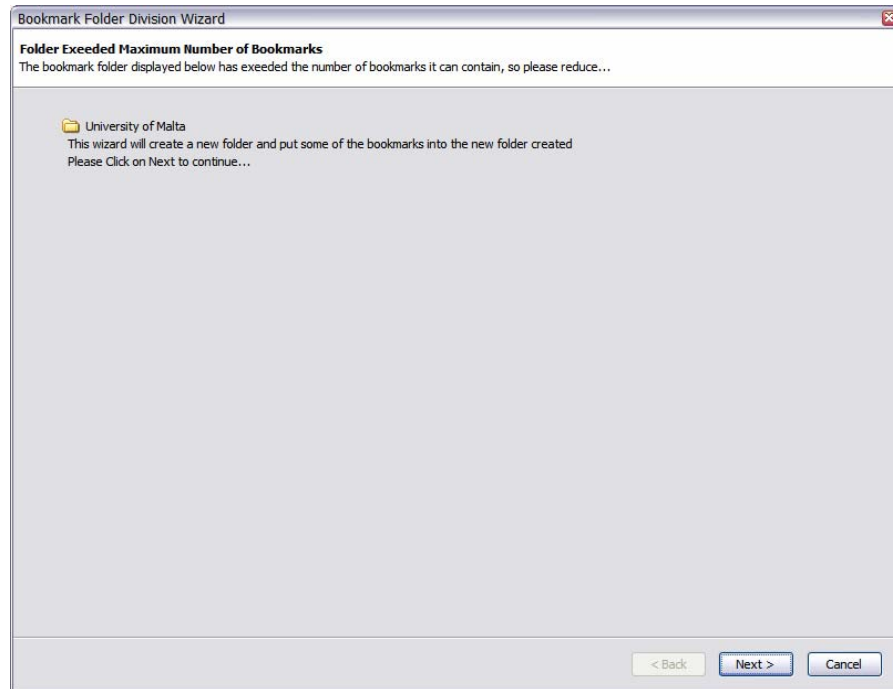


Figure 59 Divide Category Wizard Page 1

Click on Next will open the next wizard page (Figure 60) to be able of creating a new category.

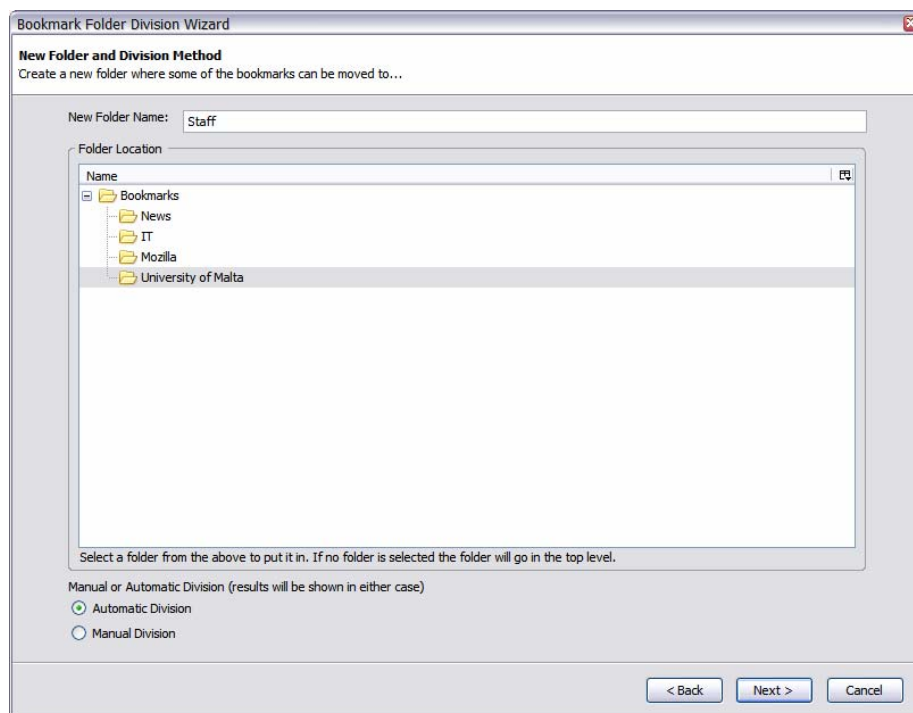


Figure 60 Divide Category Wizard Page 2

Enter a folder name and select the location where the folder will be created. You can opt for automatic or manual division. In both cases the changes can be adjusted to match your wishes.

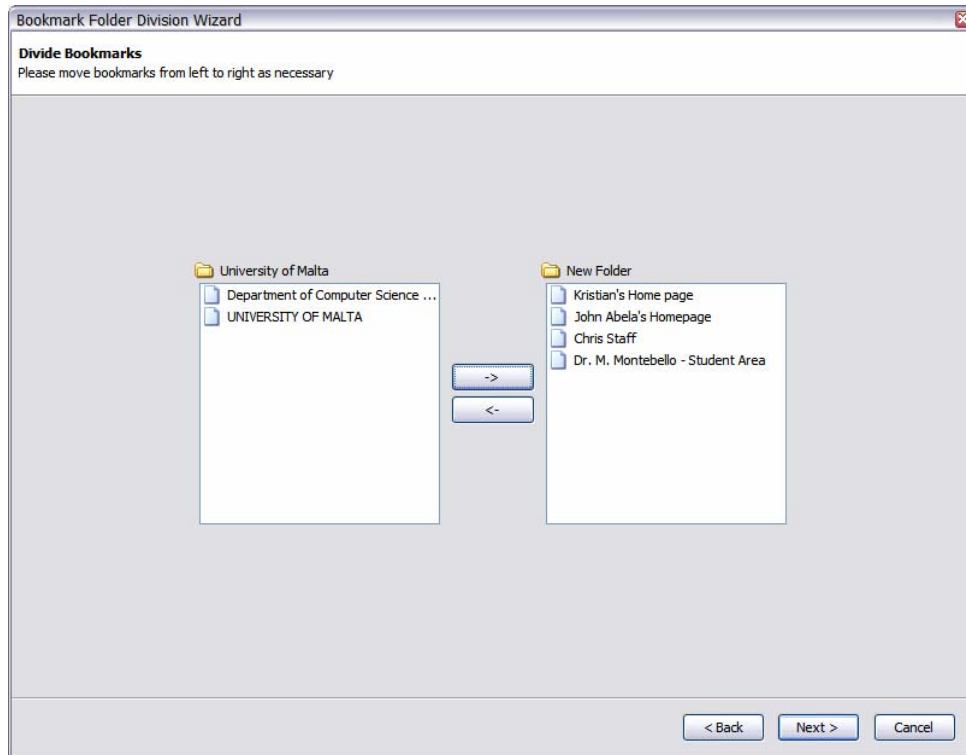


Figure 61 Divide Category Wizard Page 3

Use the → and ← arrows to move the bookmarks from the old category to the new one.

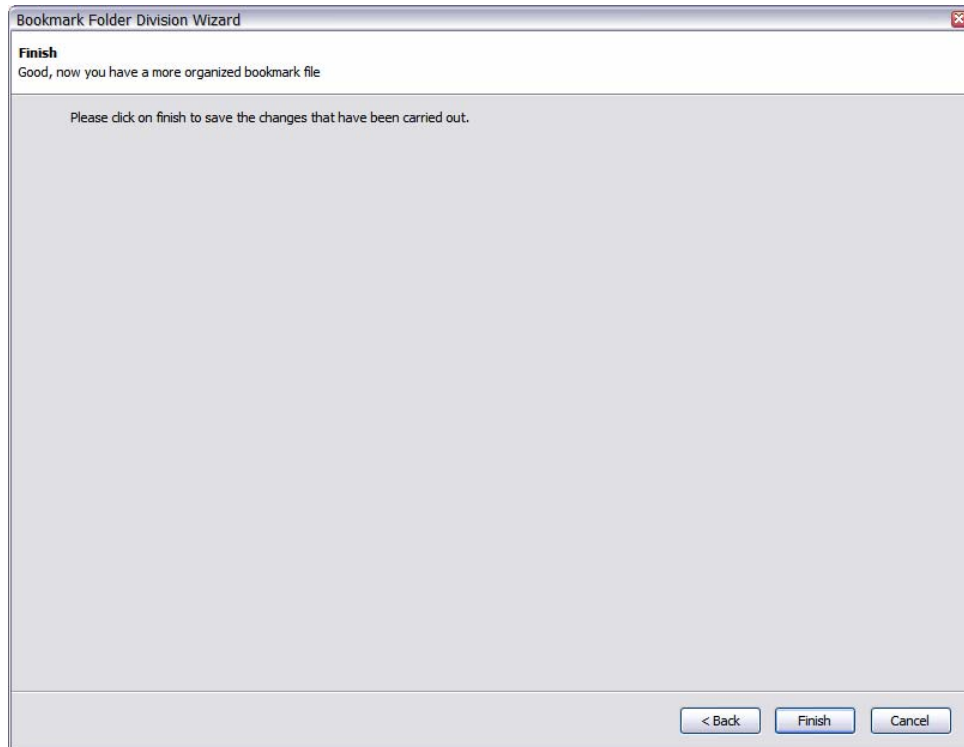


Figure 62 Divide Category Wizard Page 4

Click on Finish will save all changes done and divide the category into two new categories, having a bookmark file much more organized.

1.11 History Viewer

The history viewer window (Figure 63) shows the list of visited sites in the past 20 days (or more, according to the value set in preference. see section 1.12).

The history viewer divides the web pages visited according to the bookmark categories that one has. Search Pages results are classified automatically in a special category named "Search Pages", the rest uncategorized sites are placed in a special category named "Unclassified".

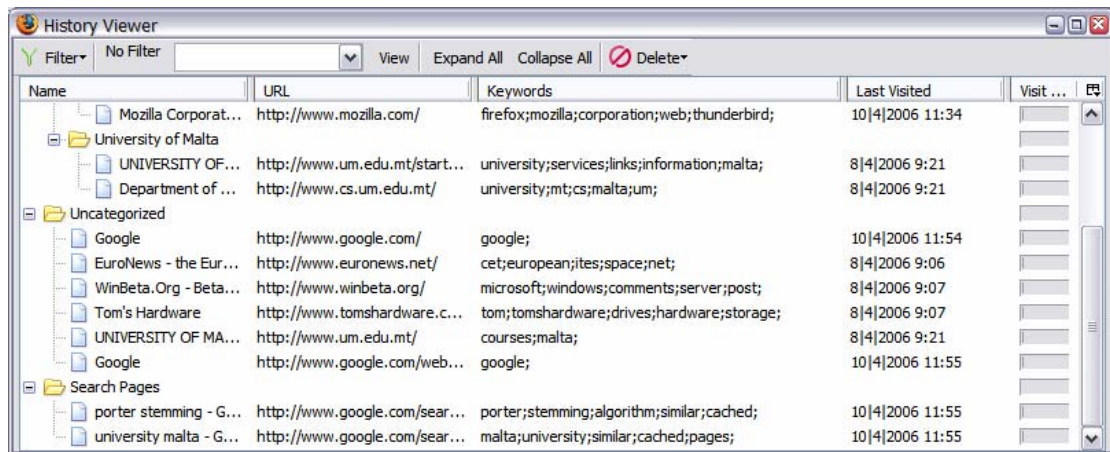


Figure 63 HyperBK History Window

There are a series of filters that can be applied to the history in order to reduce the number of entries. These filters include:

- **Date**
This filter will show all pages that were visited since the date specified.
- **Section**
Only one category is shown in the history tree.
- **Visit Count**
Filter by the number of visits to a particular page.
- **Keyword**
Search for any pages that match the given keyword. The title, page URL and page keywords are used.
- **And a special filter: Visited in Same Period**
This filter is available by right clicking on any history entry. It will filter out and show only pages that were visited 2 hours before and after visiting that particular page.

These filters can be applied by selecting the filter from the Filter menu in the toolbar and entering the parameter that is required in the drop down box. Pressing on the View button will enable the filter and showing all items that match the specified criteria.

Right clicking on a menu will display a context menu which has the options of Opening a history page, or highlighting the page Referrer. The Page Referrer refers to the page that leads into the page.

1.12 HyperBK Preferences

The preferences window opened from Tools...HyperBK Preferences has a number of parameters which can be set to make HyperBK adjust to the user's requirements.

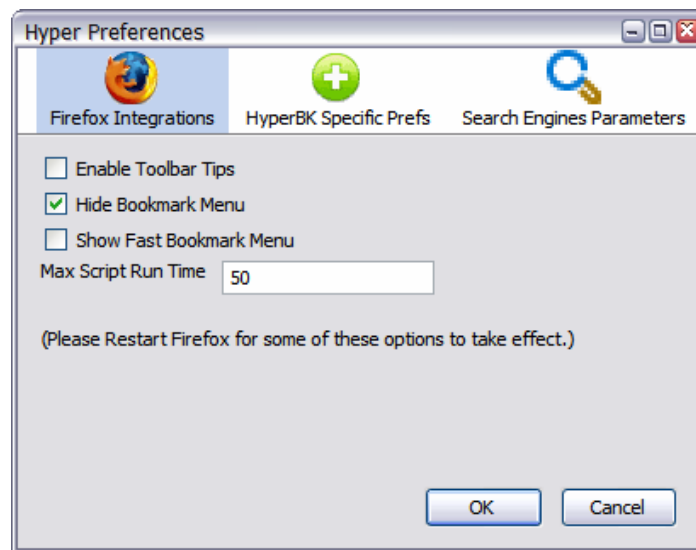


Figure 64 HyperBK Preferences Page 1

The preferences that can be set are the following:

- Firefox Integration (Figure 64)
 - Disable Toolbar Tips
Use this option to disable toolbar tips.
 - Hide Bookmarks Menu
Hide the original Firefox bookmark menu.
 - Show Fast Bookmarks Menu
Show Fast Bookmarks Menu on the window menu bar.
 - Max Script Run Time
Maximum time that a script is allowed to run.

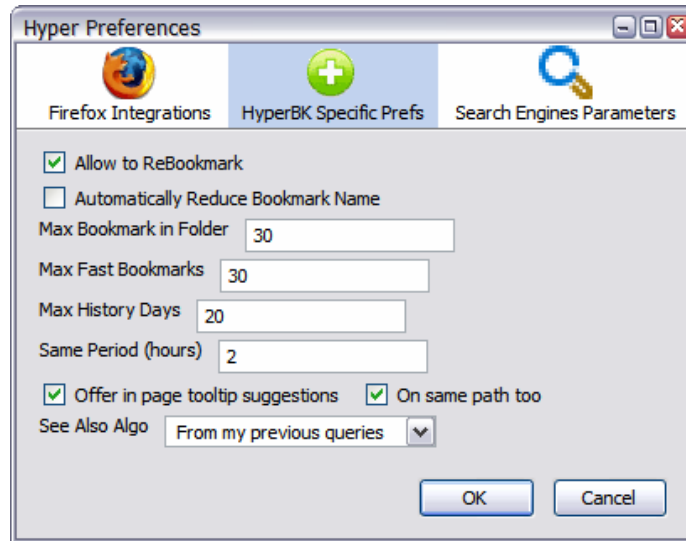


Figure 65 HyperBK Preferences Page 2

- HyperBK Specific Prefs. (Figure 65)
 - Allow to ReBookmark

Be allowed to re-bookmark once a bookmark is already present. This will not result in a new duplicate bookmark but a move and update of the already present bookmark.
 - Automatically Reduce Bookmark Name

Instead of having the bookmark name as the title this is automatically reduced.
 - Max Bookmarks in Folder

Maximum number of bookmarks that are allowed in a folder, the default is 30.
 - Max Fast Bookmarks

Maximum number of fast bookmarks that can be hold in fast bookmark list, the default is 30.
 - Max History Days

Maximum number of days to keep in history, the default is 20 days.
 - Offer in page tooltip suggestions

Offer tooltip suggestion on links by marking links that have already been bookmarked. The 'On Same Path' option will

highlight those bookmarks where a page on the same path matches (domain) but not the page itself.

- See Also Algorithm

See also algorithm to use. There are two:

- Using the SE Referrer
- Using keywords from the automatic computed query

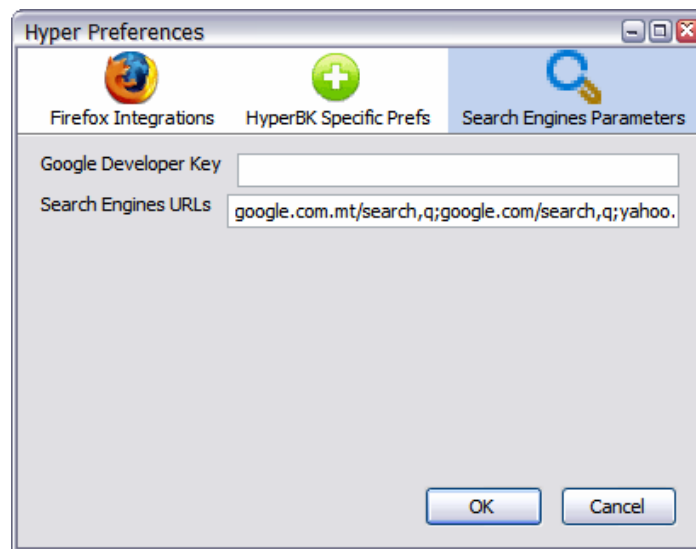


Figure 66 HyperBK Preferences Page 3

- Search Engines Parameters (Figure 66)

- Google Developer Key

This is the Google developer key which is required to use some of Google services (specifically in See Also). To apply for one please go to www.google.com/apis

- Search Engine URLs

This is a list of search engine pages URLs. The default value will match Google, Yahoo and MSN Search. It is a ; delimited string where each part consists of two pieces. The first being part of the search URL while the second delimited by ',' is the query variable.

1.13 See Also

This option will perform a search to a particular search engine and retrieve a set of URLs that are similar to the category. There are two ways this search query is computed. The first one uses the page keywords that are automatically picked out of a page while the second method uses search terms that were used to find the bookmarked pages in the first place (from the SE Referrer).

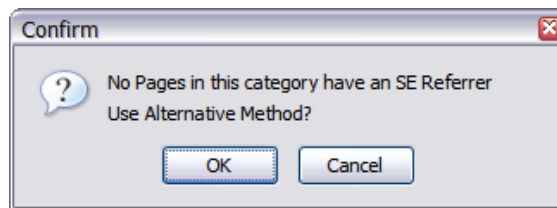


Figure 67 No SE Referrer Prompt

If none of the pages in a particular category have an SE Referrer than the method will be switched to the other automatically. If the dialog shown in Figure 67 appears, click on OK to generate a query out of the automatically picked keywords.

The automatically picked keywords will search Google directory instead of the whole WWW (Figure 68).

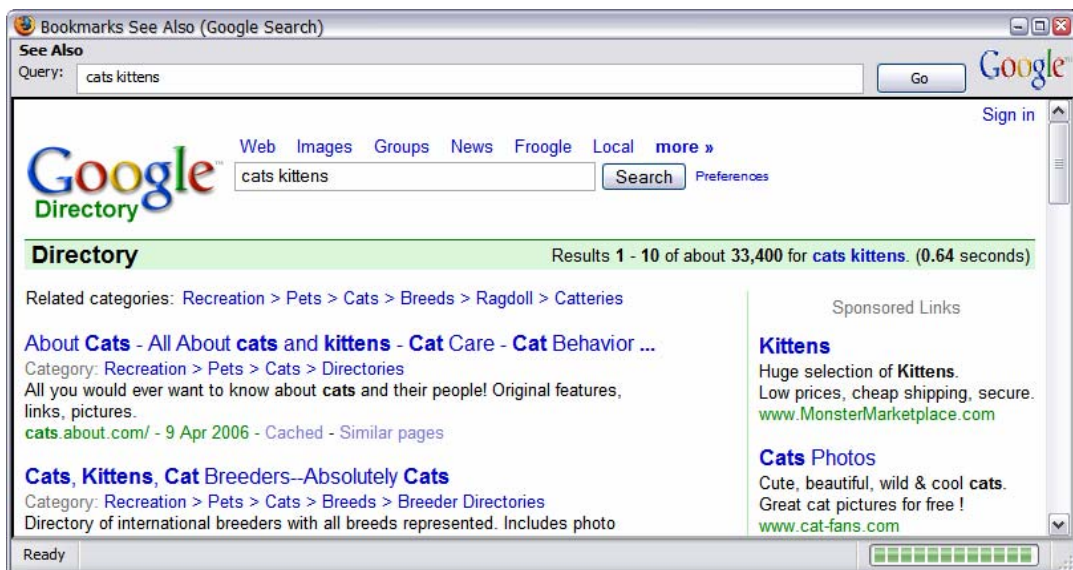


Figure 68 See Also Window (top keywords)

On the other hand the see also using the SE Referrer (previous search terms) will perform a search in the whole www as shown in Figure 69.

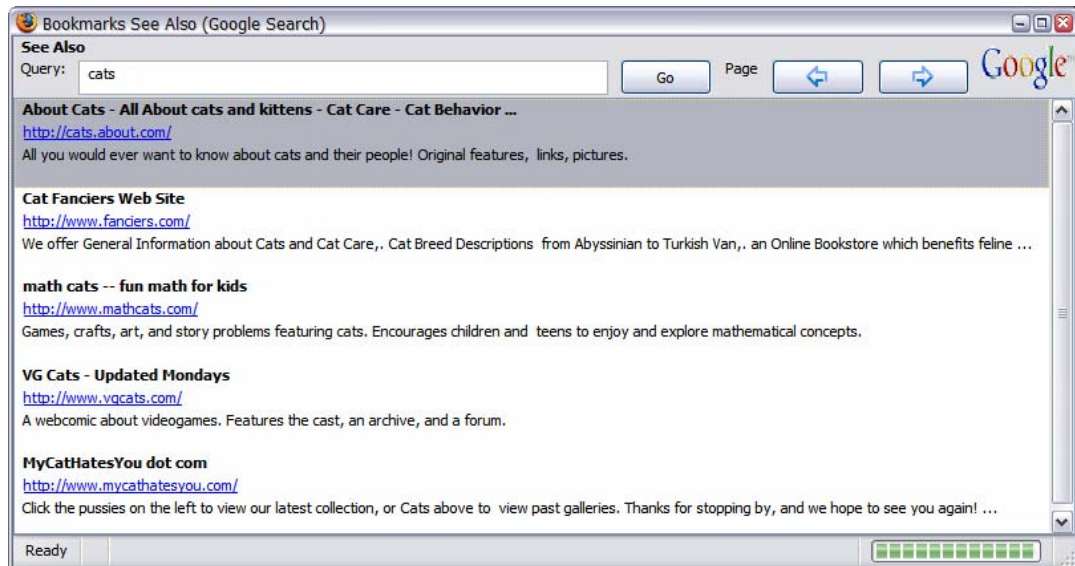


Figure 69 See Also Window (SE Referrers)

Appendix D: Contents of CD-ROM

-report	<i>Report Folder</i>
hyperBkreport.pdf	This Report (PDF Format)
hyperBkReport.doc	This Report (Microsoft Word Format)
-extension	<i>Extension Folder</i>
chrome	Chrome Folder (includes all source code)
components	Components folder (PearlCrescent)
defaults	Defaults folder (includes default preferences)
platform	Platform Specific Folder (Components)
chrome.manifest	Extension Chrome Manifest File
install.rdf	Extension Install RDF file
-other	<i>Other Folder</i>
Firefox Setup 1.5.0.3.exe	Firefox setup for Windows
-evaluation	<i>Evaluation Folder</i>
hyperbkEval	Bookmark files Collected
bkeval	Bookmark files used in evaluation
bknoteval	Bookmark files not used in evaluation
phpscripts	PHP scripts & DB (for evaluation)
-hyperE.xpi	Extension Installation Package
-info.txt	Author Details and Information