Developing an Interface Agent for a Library Website

by

Shirley Joanne Lincicum

Submitted in partial fulfillment
of the requirements for the degree of

Master of Science in Education: Information Technology

Western Oregon University

June 5, 2003
The undersigned members of the Graduate Faculty of Western Oregon University have examined the enclosed Professional Project entitled:

**Developing an Interface Agent for a Library Website**

Presented by: © Shirley Joanne Lincicum, 2003 a candidate for the degree of

**Master of Science in Education: Information Technology**

and hereby certify that in our opinion it is worthy of acceptance in partial fulfillment of the requirements for this master’s degree.

Date: ____________________

Chairperson
Examinining/writing Committee: ________________________________
Dr. John C. Marsaglia

Committee member: ________________________________
Dr. R. Dana Ulveland

Committee member: ________________________________
Dr. Mary Bucy

Director of Graduate Studies: ________________________________
Dr. Joseph Sendelbaugh
ACKNOWLEDGEMENTS

I would like to thank:

- The faculty, staff, and student workers at the Hamersly Library who showed remarkable patience with me as I struggled to balance my dual role as full-time librarian and part-time graduate student. Special thanks to those individuals who contributed to the development and evaluation of the prototype website.

- Dr. John Marsaglia, for agreeing to serve as Chair of my examination committee and for his patient guidance and attention.

- Dr. Dana Ulveland for serving on my examination committee and for his guidance throughout my course of study.

- Dr. Mary Bucy for serving on my examination committee.

- My husband, Kyle Banerjee, and my dog, Keiko, for sticking with me through the good times and the bad.
# TABLE OF CONTENTS

Acknowledgements............................................................................................................. ii  
Table of Contents............................................................................................................... iii  
List of Figures.................................................................................................................. iv  
Abstract............................................................................................................................... v  
Introduction..................................................................................................................... 1  
  Project Description...................................................................................................... 6  
Review of Related Literature.......................................................................................... 7  
Technology ................................................................................................................... 12  
The Prototype................................................................................................................ 19  
  Help Website............................................................................................................. 19  
  Frequently Asked Questions List (FAQ)................................................................. 21  
Peedy......................................................................................................................... 25  
  Evaluation.................................................................................................................... 37  
Conclusions................................................................................................................... 41  
References......................................................................................................................... 45  
Appendix A....................................................................................................................... 49  
  helpindex.htm ............................................................................................................. 50  
  index.htm ....................................................................................................................... 51  
  Peedy.htm ..................................................................................................................... 52  
  AboutPeedy.htm ......................................................................................................... 53  
  AVStutorial.htm......................................................................................................... 56  
Appendix B ....................................................................................................................... 58  
  FAQtoXML.vbs ......................................................................................................... 59  
  help.xsl ......................................................................................................................... 64  
  Circ.xml ......................................................................................................................... 66  
Appendix C....................................................................................................................... 71  
  PeedyHelp.vbs .......................................................................................................... 72  
  AVStutorial.vbs ......................................................................................................... 92  
Appendix D..................................................................................................................... 102
LIST OF FIGURES

Figure 1: Help website home page ................................................................. 19
Figure 2: About Peedy tutorial page .............................................................. 27
Figure 3: Talk to Peedy input box ................................................................. 29
Figure 4: FAQ search results display ............................................................ 31
Figure 5: WOLF keyword searching tutorial page ....................................... 35
ABSTRACT

DEVELOPING AN INTERFACE AGENT FOR A LIBRARY WEBSITE

by
© Shirley Lincicum, 2003
M.S. in Education: Information Technology
Western Oregon University
June 14, 2003

The social interaction that occurs in the traditional library setting is a unique and important component of the services that libraries provide. With services increasingly moving online, libraries face the challenge of continuing to provide a socially rich environment for their users. Interface agents are one of several technologies that have the potential to improve the sociality of library websites. This project explored the feasibility of developing an interface agent application for a library website using freely available software tools and broadly supported web standards. This paper discusses design issues and technical challenges associated with the development of agent-based applications intended to provide direct assistance to library end-users and describes the development of a prototype help website that features an interface agent, a frequently asked questions list (FAQ), and several tutorials.
Introduction

The primary goal of library services is to help people fulfill their information needs. Libraries have traditionally offered a number of services in support of this goal, such as collecting and preserving information resources, and providing an infrastructure that facilitates the location and use of resources in the collection. Especially in recent times, libraries have also offered personal assistance to support users as they search for resources to fulfill their needs. This service is usually called reference service.

Bopp (1991) describes three basic types of reference service typically delivered in libraries. Information service involves locating specific pieces of information or information resources relevant to the user’s information needs. Instruction involves teaching individuals or groups how to locate and use the information resources to which the library provides access. Guidance generally involves providing in-depth counseling or consulting for individuals in the selection and use of information resources. In most libraries, information service is the most frequently encountered form of reference service. Information service is typically delivered by professional and paraprofessional library staff stationed at publicly accessible desks within library buildings. Library users typically initiate this type of service by approaching the reference desk and posing a question. Especially in academic libraries, a certain amount of instruction is often delivered in conjunction with delivering information services to users who have approached the reference desk. Many academic libraries also offer instruction and guidance services that are scheduled in advance and take place away from the reference desk, but providing information service at the reference desk is often the library’s top...
priority. This means that academic libraries often devote the majority of the resources available for delivering reference service to providing information service.

“An information search is a process of construction which involves the whole experience of the person, feelings as well as thoughts and actions” (Kuhlthau, 1991, p. 362). Because providing direct support to users is the primary focus of reference service, reference librarians must be prepared to address both users’ information needs and their human needs as individuals who are engaged in a search for information. Research has shown that the uncertainty experienced in the initial stages of the information search process often causes significant discomfort and provokes feelings of anxiety that can adversely impact users’ ability to articulate their information needs (Kuhlthau, 1991).

Today, these feelings are often exacerbated by information overload, which leaves users feeling overwhelmed and at a loss to know where to begin looking for information. Many users are also confused and even intimidated by the technology that is now commonly used to provide access to information resources. One of a reference librarian’s primary goals is to help users overcome their discomfort and anxiety and gain confidence in their ability to make effective use of information resources. Like coaches or teachers, reference librarians are there to help users succeed. This often means providing social support in addition to reference service.

In the course of providing information service, reference librarians often engage users in a reference interview. The primary purpose of the reference interview is to provide the librarian with sufficient information to successfully satisfy the user’s information need. In practice, the reference interview consists of a conversation between the librarian and the user. Conversing with the reference librarian is often as valuable
for the user as it is for the librarian. Because many users are not fully aware of their information needs when they approach the reference desk, the act discussing their needs with the reference librarian often helps them to clarify their needs in their own minds. This can help users to progress through the information search process. Even if users are reluctant to approach a reference librarian, they often find peers with whom to converse at the library. This is especially true in an academic context, since the library is often a place where students congregate. Despite persistent stereotypes about shushing librarians, libraries are social places. Indeed, librarians would never have gotten a reputation for shushing if users had always remained silent when visiting the library.

Increasingly, users access library services through the web. This is good in that it allows users to search for and use resources from the comfort of their own homes or offices, at times that are most convenient for them. But by never setting foot within the library building, many users are missing out on the social dimension of library services. Few library websites today support a level of social interaction comparable to that found within the physical library building. Ackerman (1994) argues that even in digital libraries, which are often not associated with any physical library, it is important to incorporate features that support social interaction because of the valuable role that it plays in the process of seeking information.

With more services available online, libraries face the challenge of continuing to provide a socially rich environment for their users. In recent years, librarians have begun to focus much more attention on delivering live reference service through their websites. This service is often called digital reference. Some libraries offer digital reference using nothing more than mainstream technologies, such as e-mail or live web chat. Others are
experimenting with the use of specialized digital reference software that supports teleconferencing and other features that allow librarians to control users’ computers remotely (Tennant, 2003). Providing digital reference service is a challenging task. The technical challenges alone are significant. For example, it is difficult to conduct an effective reference interview using an asynchronous technology such as e-mail, and more sophisticated applications often suffer from glitches caused by software incompatibilities. In addition, few libraries have the personnel necessary to provide live digital reference in addition to regular reference service for a substantial number of hours every day. Staffing live digital reference 24 hours a day, seven days per week is simply out of the question for most academic libraries. The challenges associated with providing live digital reference service have led some to question whether it is worthwhile for libraries to even attempt to provide this type of service (Tennant, 2003). The push toward implementing digital reference services indicates that librarians are concerned about making core library services accessible to all users, whether or not they actually visit the physical library.

Most academic libraries today invest considerable resources in developing and maintaining websites that describe the library and its services, and provide access to an array of databases and other electronic information resources. Aside from providing access to digital reference services, however, these websites generally provide little support for social interaction. There are many ways that libraries can make their websites more socially engaging places. Providing tools such as chat rooms and bulletin boards where users can interact both with librarians and with one another seems an obvious and fairly low-tech option that few libraries have implemented to date. Because social
interaction has traditionally been an important aspect of library service, incorporating features that support social interaction into library systems has the potential to improve online library services. Interface agents are one of several technologies that have the potential to improve the sociality of online library services.

Currently, the term software agent is used in two related, but distinctly different ways. Some use the term to refer to computer programs that are capable of initiating actions, communicating with other programs, and responding to events without direct human control. These applications typically employ artificial intelligence technology in order to support these advanced features. Others define software agents more broadly as computer programs that portray themselves to the user as having human characteristics, such as knowledge and intelligence, regardless of whether the programs are actually able to function autonomously. This second definition is often called the agent metaphor (Erickson, 1997).

An interface agent is a specific type of software agent that uses a character to interact with users in a socially engaging manner. The primary goal of an interface agent is to make computer use more comfortable for the human end-user. Some interface agents also attempt to assist users in accomplishing computer-based tasks or act on behalf of the user in the online environment (Laurel, 1997). Interface agents are often capable of using multiple cues to communicate, including written text, speech, facial expressions, and/or body language. Because interface agents typically fall under the agent metaphor definition of software agents, they focus on mimicking human characteristics and can be built using fairly conventional programming techniques. More sophisticated interface
agent applications may employ artificial intelligence, machine learning, and/or natural language processing technologies, however.

**Project Description**

The primary goal of this project was to assess the feasibility of developing an interface agent application for a library website. A secondary purpose was to explore functional requirements, design issues and technical challenges associated with developing an agent-based application intended to provide assistance to library end-users. The final product of the project is a prototype help website that features an interface agent named Peedy, a frequently asked questions list (FAQ), and several tutorials.

One ancillary goal of the project was to explore the capabilities of broadly supported web standards and freely available software development tools. To date, library software vendors have shown little interest in developing applications that support the development of agents. Though a few commercial software packages designed to support the development of interface agents currently exist, none of these applications are yet in mainstream use in libraries and it is unclear to what extent these applications would support the development of agents suited to the peculiarities of the library environment (Balleste, 2002; Miller, 1999). Exploring the capabilities of freely available software tools allowed me to test the capabilities of these tools and to reduce the level of financial risk associated with the development of the prototype website.
**Review of Related Literature**

Pioneering research reported by Reeves and Nass (1996) indicates that with even the slightest of cues, people respond to media in a social manner. These responses apply both to appliances that deliver media and to characters that appear on the screen. People seem to instinctively apply social rules in making judgments when interacting with media, including computers. For example, in a series of experiments, Nass and colleagues observed that people seem to obey rules of politeness and expect computers to reciprocate, even when they assert strongly under direct questioning that they view computers as nothing more than inanimate machines. When people in these experiments felt that computers had either ignored or violated politeness rules, their interactions with computers became noticeably more uncomfortable and unpleasant.

Reeves and Nass (1996) refer to the model derived from their research as the media equation. This model is important to the development of human-computer interfaces for two reasons. First, it provides valuable insight into why people feel the way they do when they use computers. According to the media equation, all computer interfaces have social effects because people are conditioned to respond to computers in a social manner. Sophisticated representations are not necessary to convey personality in computer interfaces. People will perceive personality in every interface, whether it has been designed to have one or not. Many current applications routinely violate common social rules and trigger negative responses in users because they were designed without much regard for the social dimensions of human-computer interaction. Second, the findings provide strong evidence to support the design of effective interface agent.
applications and the hypothesis that these agents can be used to improve the sociality of human-computer interfaces.

Several aspects of the media equation have particular relevance to the design of interface agents. People tend to form social bonds when teamed with either humans or machines to accomplish a task. This suggests that if people can see computers more as peers, feeling somewhat dependent on the computer, but not too inferior or superior, they might have more productive relationships with computers. Personality is socially powerful, so especially when using characters in an interface, designers must use great care to use a personality type that is appropriate to the role that the character is intended to play within the application. The character’s personality must also be communicated consistently throughout the application because inconsistent personalities make people uncomfortable. Because people like it when others change to suit their preferences, adapting the personality of an interface agent to match that expressed by the user can be a good way to personalize an interface. People like to hear praise, but ascribe more intelligence to those who criticize, so there is a trade-off between perceptions of likeability and intelligence when designing interface agents. Because people automatically assign responsibility for messages to those who deliver them, people will often perceive interface agents as the source of the information they deliver (Reeves & Nass, 1996).

Empirical research into the effects of interface agents is still in its infancy, so few firm conclusions have yet been drawn. In their review of recent research regarding the impact of interface agents, Dehn and Van Mulken (2000) found that though current findings do not provide sufficient evidence to conclude that systems with interface agents
are generally superior to those without such agents, there is evidence that interface agents have positive effects. For example, research indicates that users find systems with agents more entertaining than systems without agents. Moreno, Mayer, Spires, and Lester (2001) found that students using multimedia instructional software learned more deeply when the application included an interface agent than when it contained only text and graphics. Moundridou and Virvou (2001) found that using an interface agent in an online tutoring system enhanced students’ learning experience, even though it did not significantly improve learning outcomes.

Head (1998) describes the development of interface agents for Microsoft Office (Office). Usability testing conducted by Microsoft prior to the release of Office 97 revealed that many users, even experienced ones, were unaware of much of the functionality included in the software. In an effort to increase the usability of the software and reduce the number of calls to its telephone support service, Microsoft poured considerable resources into designing and testing interface agents, called Office Assistants, that were first incorporated into Office 97. Office Assistants serve as a flexible, central source of help. They offer suggestions and draw the user’s attention to information provided in the online help file, rather than simply completing tasks on the user’s behalf, so that users are more likely to increase their own skills. Though the software monitors the user’s environment and prompts the Assistant to offer help proactively, the software uses little artificial intelligence technology. This limits the Assistants’ ability to diagnose problems correctly and converse with the user, but when they get it right, Office Assistants can significantly improve the quality of the interaction between computer and user.
In the process of defining the requirements for software agents intended to provide expert services to users, Nardi and O’Day (1998) studied reference librarians as exemplar human agents. They found that librarians routinely create representations that place requests within a broader context of user preferences and constraints. Based on their research, they do not believe that it is possible to design agents capable of providing the same level of service that human librarians do, but they argue that well-designed software agents can be used to personalize technology use. They also envision librarians and software agents working collaboratively to satisfy users’ information needs, so they argue that librarians should be actively involved in designing, organizing, and providing access to software agents of all types.

The most obvious application for interface agents in the context of library services is in providing services to users who access services online. Interface agents could be deployed as:

- Greeters or hosts, welcoming users to the library website and making announcements,
- Tour guides, providing an introduction to the services available on the library’s website, offering explanations of the various services offered and responding to users’ questions,
- Tutors, instructing users in routine tasks such as searching for materials in the online catalog or submitting an interlibrary loan request,
- Personal assistants, monitoring the user’s session and proactively offering context sensitive help and/or support in completing tasks.
One of the biggest challenges in providing online library services is providing a consistent user interface. Library websites typically tie together web-based resources from many disparate sources, including competing database vendors. Users are confronted with many purposely different interfaces as they search for information. A robust interface agent application could provide a familiar social presence that could accompany users throughout their visit to the library’s website. Library websites must also compete with other websites, such as Google, for users’ attention and business. Adding interface agents to the library website could help the library to attract and retain users’ attention by providing an environment that is more welcoming and personalized than that they may find elsewhere. Finally, interface agents could be used to supplement the services provided by human reference librarians, allowing libraries to provide some level of socially engaging service through their websites 24 hours a day, seven days per week. If users begin to rely more on interface agents to answer routine questions, libraries might also be able to devote more of their resources to providing higher level reference services, such as guidance services.
Technology

For a number of technical and practical reasons, I chose to focus on client-side development in this project. In the web environment, computers fall into two fundamental categories: servers and clients. Servers are remote machines to which users connect in order to access web-based content. The user’s personal computer is the client. Client-side applications are programs that may be downloaded from a server, but actually execute on the client computer. From a technical perspective, client-side applications often offer greater responsiveness for the end-user. Instead of having to send requests and responses back and forth over the network, a client-side application allows the user to download both data and programs capable of manipulating the data all at once. This also has the desirable effect of reducing the load on the server. Focusing on client-side development also allowed me to minimize the complexity of this project. Developing programs that run on the client allowed me to experiment with new technologies without much concern about compromising the security of the campus network. I was able to accomplish the entire project using my existing permissions and directory on the library’s website, and without having to burden overworked network support staff with monitoring my activities and/or cleaning up the messes that I would have created had I been experimenting with server-side programming. My previous programming and web publishing skills are also better suited to client-side programming. I was eager to use this project as an opportunity to update and significantly expand my skills, and to explore the capabilities of recently issued standards intended to support the development of client-side applications.
Because producing a fully developed web application was not the main purpose of this project, cross-browser and cross-platform support was not a primary concern. The prototype was developed and tested using Microsoft Internet Explorer 6 running under Microsoft Windows XP. This is the standard software platform provided on all public and staff workstations in the Hamersly Library. The prototype website was not tested using other browsers or operating systems. Obviously, such testing will have to occur and any issues revealed will have to be addressed before any components of the prototype can be incorporated into a regular website.

The prototype application uses a combination of technologies. The web pages that provide the foundation for the site use Hypertext Markup Language (HTML) and Cascading Style Sheets (CSS). HTML is the markup language most commonly used to encode documents for display on the web. CSS is a standard that works in conjunction with HTML to specify how web documents are formatted and displayed. The FAQ uses Extensible Markup Language (XML) and related standards. XML is a subset of Standard Generalized Markup Language (SGML) that has been specifically designed to facilitate data interchange over the web. Like other markup languages, such as SGML and HTML, XML supports the creation of encoded text files that can be interpreted and manipulated by computer programs.

XML is significantly less complex than SGML, but much more flexible and expressive of document content and structure than HTML. Despite its name, XML is not really a language itself, rather it’s a standard that designers can follow in creating markup languages that express the unique structure and content of their documents (Hunter, 2001b). This makes XML very powerful, but it also makes it more complicated to work
with than a markup language such as HTML, which provides explicitly defined tags that mainstream web browsers have been pre-programmed to understand and interpret. Because XML tags are defined by authors or designers themselves, software applications must be specially programmed to interpret those tags. Increasingly, the major web browsers, especially Internet Explorer, provide some basic support for XML, but because of the way the XML specification is structured, it takes several components in addition to an XML document and an XML-aware web browser to generate user-friendly displays of XML documents. XML is designed to work with other standards, such as the Extensible Stylesheet Language (XSL) and CSS, to handle document display. The major advantage to this is that document content is not inextricably linked to instructions for display. To modify how XML encoded content is displayed to end-users, the designer need only modify a style sheet; there is no need to alter the document containing the data itself. The data contained in a single XML document can also be displayed in multiple ways simply by developing multiple style sheets. Another advantage to using XML is that instructions for processing documents can be carried out on the client, rather than the server, which can improve the responsiveness of applications and reduce server load (Hunter, 2001b). The major disadvantage to using XML is that it can increase the complexity of web publishing, but the flexibility it provides often outweighs this disadvantage.

I used Microsoft Agent (MS Agent) and Microsoft Visual Basic Scripting Edition (VBScript) to develop the animated interface agent itself. MS Agent is, “a set of software services that enables developers to incorporate interactive animated characters into their applications and web pages” (Microsoft Corp, 2003, April 2, p. 1). It is a Microsoft Windows (Windows) core component that comes pre-installed with current
versions of the Windows operating system and can be installed to run under versions as far back as Windows 95. Currently, MS Agent only operates on the Windows platform. Though use of MS Agent is subject to a license agreement, there is no fee associated with using it. Developers must complete a separate royalty-free license agreement if they wish to distribute MS Agent or any of its components from their own websites or bundled along with their non-web-based applications.

Of the services provided by MS Agent, arguably the most powerful are those provided by the animation server, which manages all aspects of character display. All the programmer has to do is make a call to a method that causes the character to move. The agent animation server takes care of all the details related to compiling and executing the appropriate character animation, including providing lip synchronization if the user’s computer is configured to support audio output (Trower, 1999). For each of the four default characters it provides, MS Agent supplies a character data file that contains all of the data necessary to support a rich collection of pre-defined animations. MS Agent also provides a Character Editor that developers can use to compile animation files for custom-designed characters.

MS Agent characters can communicate with users in several ways. They automatically display text in a word balloon as they speak. Words appear one by one, as they are spoken, and the text balloon disappears automatically a few seconds after each call to the Speak method concludes. It is not possible to display blocks of text in their entirety or to accept user input via the text balloon in MS Agent, as can be done with Microsoft’s Office Assistants. The software does provide users with the ability to control the speed of the spoken output. MS Agent also comes bundled with a text-to-speech
engine and the Microsoft Speech Recognition Engine. This is arguably one of MS Agent’s most valuable features because it allows characters to produce spoken audio output on-the-fly and to respond to voice commands if the user’s computer is properly configured to support this functionality (Trower, 1999).

The MS Agent programming interface supports a number of objects. These objects provide the methods and properties used to control the character and monitor the events that occur in its environment. The Agent Control object is the “master” object that provides access to other types of objects and to the most significant events, such as the Click event that fires when a user performs a mouse click on the displayed character. The Character object provides most of the methods and properties used to control the interface agent character. For example, this object provides a Speak method that instructs the character to speak a text string and/or play a sound file supplied by the programmer. Complete documentation for the MS Agent programming interface is available in Microsoft Agent SDK (Microsoft Corp., 2002), which is available online through the MSDN Library.

In order to avoid disruptive delays in program execution, the agent server processes some calls asynchronously. This can cause problems when the programmer wants to synchronize a particular animation or spoken instruction with another activity, such as the display of a dialog box to accept input from the user (Trower, 1999). Request objects allow the developer to keep track of the status of requests that have been sent to the server. Using request objects in combination with the Request event handlers provided by the Agent Control object allows programmers to delay the execution of certain instructions until a specific asynchronous method has executed. The programmer
can also embed bookmarks within the code and use the Bookmark event handler provided by the Agent Control object as a synchronization tool (Trower, 1999).

The MS Agent programming interface can be accessed by any Component Object Model (COM) compliant programming language. MS Agent also includes an ActiveX control that allows developers to program characters using common web scripting languages, such as VBScript and JavaScript. This is what allows developers to incorporate interface agents into web-based applications. JavaScript is used more often than VBScript because it enjoys broader support among current web browsers. I chose to use VBScript in this project primarily because I was more familiar with Visual Basic syntax from my previous programming experience. Since few browsers other than Internet Explorer currently support MS Agent, the fact that VBScript is not widely supported by other browsers was not a major concern.

The Document Object Model (DOM) and one Dynamic Hypertext Markup Language (DHTML) property are used in the scripts that support Peedy. The Document Object Model (DOM) is a collection of platform and programming language neutral application programming interfaces developed by the World Wide Web Consortium (W3C). It provides a core set of properties and methods that can be used to access and manipulate structured web documents (Marini, 2002). The DOM is implemented widely, if somewhat inconsistently, in most current web browsers. The main disadvantage of using the DOM are its resource requirements. In order to process documents, the DOM must load the entire document into memory. This can be a major drawback when working with large files (Hunter, 2001). DHTML is a name given to a collection of extensions to HTML implemented by Microsoft. DHTML combines features from a
number of open standards, including HTML, CSS, and the DOM, with proprietary features developed by Microsoft. The stated purpose of DHTML is to support the development of web pages that can be dynamically modified through client-side scripting or other similar techniques (Microsoft Corp., 2003). Microsoft has proposed that DHTML be adopted as a W3C standard, but this has not yet been done.
The Prototype

Help Website

http://www.wou.edu/provost/library/staff/lincicum/helpindex.htm

The help website home page consists of a frameset containing four frames (see Figure 1 and helpindex.htm in Appendix A). The larger frame on the left-hand side of the window serves as the main display frame. The initial page displayed here is the main index page for the site (see index.htm in Appendix A). At this point, this page only contains links to subsidiary help pages developed as part of this project, including the FAQ pages and the Wolf Keyword Searching tutorial page (see AVStutorial.htm in

Figure 1: Help website home page
Appendix A). A single style sheet is used to provide a consistent look for the index page and all subsidiary pages in the site. When the user clicks on one of the links on this page, the new page replaces the index page in the main display frame. Within the site, all links to external pages have target attributes that direct display to a separate window. This allows users to keep their place within the help site even as they browse freely through external web pages.

The smaller frame on the right-hand side of the window displays Peedy’s home page (see Peedy.htm in Appendix A). This page hosts Peedy. An <object> tag on this page creates an MS Agent client application and launches the agent server automatically when this page is loaded. If the user views the site using Microsoft Internet Explorer version 3.02 or later and MS Agent is not yet installed on the user’s computer, the browser will give the user the option to automatically download Agent from the Microsoft website. The MS Agent server remains loaded only as long as there is at least one client application maintaining a connection. This means that if the user closes the browser window displaying the help frameset, the agent server will automatically shut down and Peedy will disappear. Reloading either Peedy’s home page alone or the entire frameset reinitiates the client application. This behavior was one of the major factors driving my decision to use frames in the prototype website, since using frames allows a single instance of Peedy to continue running even as users browse freely through other pages. Early in the design process, I experimented with using a pop under window to achieve the same result. This worked, but when I compared the two options, I felt that using frames would provide a more unified interface environment for the end-user.
Despite all that is going on behind the scenes when Peedy’s home page loads, the page itself appears very spartan. It simply contains a text input box and several buttons that allow users to control and communicate with Peedy. I used a subtle background color to provide some visual definition within the frameset display. The top and bottom frames contain the library’s standard banner and footer. Target attributes added to the links on these pages launch a new window if a user clicks on a link.

**Frequently Asked Questions List (FAQ)**

Designing the frequently asked questions list (FAQ) was among my first challenges in the design process. Though the library’s website contains a significant amount of information intended to help patrons, this information has not been systematically gathered together and presented under a unified interface. I started by collecting existing content from various pages on the library’s site and informally logging questions that I encountered frequently during my own shifts at the reference desk. I also informally polled the student workers who staff the library’s information desk and reviewed FAQs maintained by other libraries to gather ideas. To keep this part of the project as manageable as possible, I focused on simply creating a representative set of questions and answers that could form the basis of a comprehensive FAQ. I reasoned that production of a comprehensive FAQ could wait until after library staff had a chance to evaluate the prototype.

To experiment with different ways of structuring the FAQ data, I started inputting data into a Microsoft Excel (Excel) spreadsheet. The design evolved into a fairly simple spreadsheet with seven columns, one for each data element associated with an entry in the FAQ. The seven data elements include: Question, Answer, Category, Anchor,
Keywords, See also, and Tutorial. The Question element contains a topical heading for each entry, phrased in the form of a question. I did not use actual questions, rather I composed questions representative of those typically asked about a given topic. For each Answer element, I composed a brief response to each question, including links to other pages on the library’s website when appropriate. At first, I attempted to use Excel’s hyper linking functionality to embed these links, but this allowed me to input only one hyperlink per cell. I ended up simply embedding raw HTML anchor tags (<a>) in my answer text to represent the links. This allowed me to embed as many hyperlinks into the answer text as was appropriate to the given question. I divided the questions into four broad categories: Reference, Interlibrary Loan, Circulation, and Vocabulary. At first, I recorded the category to which each question was assigned in the Category element. As the amount of data in the FAQ grew, however, I decided to create separate worksheets within the Excel file for each of these categories. This made it easier to manage and browse the data. I retained the Category element in order to group the entries into sub-categories, which I simply established as I went along. At this point, my application does not make extensive use of the Category element, but I anticipate that this element will become more important as the amount of data grows. The Anchor element contains a unique identifier for each entry that is used to support internal linking within each FAQ page when they are rendered in HTML. The Keywords element contains the searchable text associated with each entry. The See Also element is intended to contain references to other entries in the FAQ. The Tutorial element is intended to contain references to tutorials appropriate to the topic addressed in the question. Neither of these final two
elements is used in the prototype application, but I have retained them in the data
structure due to their potential for future use.

One of my primary concerns in developing the FAQ was to make it as easy as
possible to expand and update the content over time. I wanted to keep structural data
elements separate from display elements. This makes it easier to modify and enhance the
presentation design over time than it would be if the source data were simply
incorporated into static HTML-encoded web pages. In the event of a major graphical
redesign of the library’s website, for example, migration of the FAQ data will be greatly
simplified by separating content and display elements. It also provides for consistent
presentation formatting throughout the FAQ documents and relieves the individuals
responsible for maintaining FAQ content of the burden of having to also worry about
tagging the content for display in a consistent manner. I wanted to provide a data entry
and editing interface familiar to the majority of the library staff. In order to keep things
as simple as possible, I decided to retain the Excel spreadsheet that I had initially
developed. Since most library staff regularly use Excel spreadsheets, most are familiar
with its interface and with entering simple textual data into worksheet cells. This allows
those contributing content to focus on the content, rather than having to learn how to use
a new software application or text mark-up language. Excel also provides valuable data
manipulation and editing features, such as sorting and find and replace.

In order to encode the data stored in the FAQ spreadsheets in XML, each of the
four Excel worksheets is saved as a tab-delimited text file. XML documents are
generated from the delimited text files by a script written in VBScript (see
FAQtoXML.vbs in Appendix B). The script simultaneously generates five separate XML
documents from the data files, one for each of the individual worksheets and a single file
containing the data from all four worksheets (for an example, see Circ.xml in Appendix
B). Each XML document has the same basic structure. The root element, <document>,
contains a child element called <entry> for each row in the spreadsheet. Each <entry>
element contains seven child elements, one for each column in the spreadsheet:
<question>, <answer>, <category>, <anchor>, <keywords>, <seealso>, and <tutorial>.
Within the <answer> elements, one or more child elements named <a> occurs whenever
a hypertext link has been included in the answer data. The <a> element contains one
attribute, href, that contains the URL associated with the hypertext link.

An XSL style sheet is used to extract and convert data from the XML files to
HTML for display on the website (see help.xsl in Appendix B). Initially, I hoped that the
HTML tags that I had embedded within my answer elements would simply display as
inline links when the FAQ data was displayed. Unfortunately, this was not the case
because the Microsoft XML parser interprets these data elements as XML children of the
<a> tag. In the end, I was unable to determine how to create inline links using the
XSL style sheet, so I simply created a special template that extracts the <a> elements and
constructs links that display immediately following the answer text.

I considered creating a database for the FAQ, but decided against this for several
reasons. First, I had more experience working with Excel and XML than I did with
designing web-based databases. I had previously written scripts to create and manipulate
delimited text files, so writing a similar script to convert the raw FAQ data to XML
format for this project was reasonably easy. Designing and implementing a web-
accessible database would have involved a great deal more time and effort on my part,
which I didn’t feel was really necessary given the limited size of the data files with which I would be working. I also wanted to make sure that building a user-friendly interface for FAQ maintenance did not absorb a disproportionate amount of my time. I was careful to structure the data in Excel in a way that would allow it to be exported to a database at some point in the future, if desired.

**Peedy**

Peedy the parrot is one of the four default characters provided by MS Agent. I chose him because among the available options, he best suited my design goals. Peedy’s general appearance is friendly, and his animations are imbued with subtle humor that cleverly exploits the stereotypes that people associate with parrots. Peedy’s personality is fun and friendly, making him a good choice for a character intended to help end-users. Within the context of my application, there is a reasonably good match between user expectations and Peedy’s programmed capabilities. For example, people know that parrots can talk, but they do not expect parrots to possess human intelligence. MS Agent offers an option that developers can program to allow users to select their favorite character from a list of options, but to minimize the complexity of the prototype application, I chose not to offer this functionality. Such functionality could be added in the future.

PeedyHelp.vbs (see Appendix C) contains all of the code that supports Peedy’s core functions, as well as most of the code associated with the About Peedy page. I implemented several of the standard MS Agent event handlers to control how Peedy responds to certain events, such as the user clicking on him or dragging him to a different location on the screen. I also added several commands that are accessible via Peedy’s
popup menu, which appears when the user right-clicks on him. Among the commands I added are the “Stop” command, which allows the user to simply stop Peedy in his tracks, and the “Repeat last statement” command, which causes Peedy to repeat the last line of dialogue that he uttered. I also added an “About Peedy” command, which loads the About Peedy page into the main display frame. This page provides short tutorials explaining how to control Peedy, and it is the same page that loads when the user clicks on the About Peedy button on Peedy’s home page (see Figure 2 and AboutPeedy.htm in Appendix A). I wanted to make sure that users had easy access to this page at all times.

PeedyHelp.vbs runs automatically when Peedy’s home page loads into the help website frameset. The script first initializes global variables and sends requests to download Peedy’s character file and the specific animations used in the script from the Microsoft website. Because the Load and Get methods execute asynchronously, I use request objects to capture the status of these requests. This allows me to use the Agent_RequestComplete event handler to postpone execution of Peedy’s introductory subroutine until after all of the requested components are fully loaded. This is especially important over a slower connection, such as a modem connection. For the purposes of the prototype, I did not write extensive routines to handle errors that might arise if one or more MS Agent components is not properly installed on the user’s computer. This functionality will need to be added to the script if it is incorporated into the library’s regular website.
Once the character file and animations are successfully loaded, the DoIntro subroutine executes. This subroutine first calls the subroutine that initializes the menu of controls available when the user right-clicks on Peedy. It then moves Peedy to a specific position on the screen and displays him using the Show method. Next, it executes a series of calls to the Play and Speak methods that specify the animations to play and the lines that Peedy speaks as he greets and introduces himself to the user. In his dialogue, Peedy instructs the user to click on him in order to introduce himself. Within the script, I use a global variable “click” to store an integer value that is tested in the Agent_Click event handler. In this subroutine, the global variable “click” is set to 0 so that a dialog box to accept the user’s first name as input will appear when the user left-clicks on
Peedy. Upon completion of the DoIntro subroutine, Peedy enters an idling state until the user initiates an event that calls Peedy to perform another action. The initial idling animation displayed is specified in the last instruction in the subroutine, but after a certain amount of time has elapsed, the MS Agent animation server will automatically take over and cycle randomly through Peedy’s various idling animations. This is a nice feature since it provides enough visual variation to remind the user of Peedy’s presence without being particularly distracting.

Once execution of the DoIntro subroutine completes, Peedy’s next move is under the user’s control. If the user does as Peedy instructed and left-clicks on him in order to introduce herself, the Agent_Click event handler is triggered. The instructions executed from here depend upon the values associated with several variables. First, the script checks to see which mouse button was clicked. This value is automatically passed to the event handler. If the left mouse button was clicked, the handler executes instructions based on the value of the “click” variable. In this case, Peedy will prompt the user to enter her first name as the script calls the GetName function. This function displays a dialog box to accept the input from the user and calls the NameProcess function to examine the text string input by the user.

Because the script will store the data input by the user for the duration of the user’s session so that Peedy can use it in his dialogue, the NameProcess function tests the data against some rules and attempts to detect and correct for some common errors. For example, Peedy asks the user to enter only her first name. If she enters more than three words in the box, the script will extract the first word and call Peedy to ask the user if it’s o.k. for him to use this word as her name. If the user answers yes, the script will assign
the word to the “UsrName” global variable. If the user answers no, she will be given another opportunity to enter her first name. If the user clicks cancel or simply fails to enter any data when presented with the name input dialog box, the script will simply set the “UsrName” variable to nothing, and Peedy will not use a personal name when conversing with the user for the duration of the session. Once the “UsrName” variable has been set, the script sets the value of the “click” variable to the default value of 1.

Now, when the user left-clicks on Peedy, the script will prompt Peedy to acknowledge the user’s action and call a subroutine that displays a VBScript input box for accepting user input (see Figure 3). The user can also enter data by typing text into the text box on Peedy’s home page and clicking the “Ask” button. The script uses the same functions to process user input, regardless of how the user chooses to enter the data.

Figure 3: Talk to Peedy input box
Peedy’s script contains several routines that handle processing of user input. If the user enters nothing, or clicks on the cancel button provided on the VBScript input box, the script prompts Peedy to respond immediately. If the user enters some text, the subroutine calls other routines to examine the input. First, the input is passed to the Sanitize function, which checks the input text against a list of vulgar words that are unlikely to be used in a legitimate query or a polite conversation with Peedy. If any such words are found in the user’s input text, the VBScript Replace function is used to replace the offending words with the word “bleep” in the user input string. This allows the script to process any other, more polite words included in the user’s text input, so that Peedy can respond in a positive and polite manner, no matter how rude the text input by the user might be. This is consistent with conventional expectations for good customer service behavior. It also allows for the possibility that a false match could have been made by the text processing algorithm, which is entirely possible given its current level of sophistication. Once the text input by the user has been sanitized, the processed input string is sent to the Analyze function for further processing. The Analyze function examines the input string in order to detect specific characters or phrases that indicate something about its meaning. First, it looks to see if there is a question mark appended to the end of the string. If there is, it sets an integer variable to indicate that a question mark was detected. Next, it looks for specific phrases, such as “how are you” and “joke”, and sets a second integer variable if any such phrases are detected. Once all tests have been performed, the input string and the two integer variables are passed to the Respond function.
The Respond function uses the values of the variables to determine how Peedy responds to the user. If one of the pre-defined phrases was detected, Peedy responds conversationally with dialogue and animations appropriate to the assumed situation. For example, if the script detects the word “joke” in the input string, it calls the TellJoke subroutine, which prompts Peedy to tell a joke that is selected at random from those included in this subroutine. If the script fails to detect any of the pre-defined phrases, the input string is passed to the Find function, which handles the process of searching the FAQ and displaying any entries that might be relevant to the user’s query (see Figure 4). References to a question are included in Peedy’s dialogue only if the question mark variable indicates that a question mark was detected in the user’s input.

Figure 4: FAQ search results display
The actual work of searching the FAQ is done by the MatchInput function. Called by the Find function, it takes as input the text string input by the user. This text string has already been processed by both the Sanitize and Analyze functions. It sends the input text string to the Extract function, where each word in the string is compared to a list of stop words. The Extract function removes any stop words it detects and returns the modified text string. The VBScript Split function is used to convert this string to a string array in order to provide access to the individual terms. Four different types of XML objects are then created. Using the DOM, FAQ.xml is loaded into memory and references are established for the <keyword> and <anchor> elements, which will be used by the search algorithm. Results.xml is loaded next. This document provides a template into which entries extracted from FAQ.xml will be inserted as matches between terms in the FAQ <keyword> elements and the terms input by the user are detected. This document will only exist in memory; the original Results.xml document stored on the server will not be modified. The remaining two XML objects are the XSL style sheet that provides processing instructions and the XSL Template object that does the actual work of applying the XSL style sheet to the modified Results.xml document so that it can be displayed to the user in the main display frame of the help website.

The search algorithm reads the terms contained in each <keyword> element into a string array. Nested For loops are used to compare each term in the array containing the terms input by the user with the keyword terms. If a match is found, the integer variable “rank” is incremented by 1. After all of the keyword terms for the current FAQ entry have been compared, the “rank” variable is compared to the “highrank” variable, which contains the highest number of matches between the user’s terms and the keyword terms.
found thus far. If the current comparison found more matches than the previous high, "highrank" is assigned the current value of "rank" and the text contained in the <anchor> element of the current FAQ entry is recorded in the string variable "strAnchor", overwriting any data previously stored there. If the current value of "rank" is equal to the value of "highrank," the text from the <anchor> element is appended to the existing data stored in "strAnchor".

Once the keywords from each entry in the FAQ have been compared to the user’s terms, the value of “strAnchor” is examined. If this variable contains a blank string, this means that no matches between the user’s terms and the FAQ keywords were found. A brief “not found” message containing the user’s search terms is constructed in HTML and passed to the Render subroutine, where it is written to the screen in the main display frame using the DOM document.write method. If “strAnchor” contains data, the value of the text in the anchor element for each entry in the FAQ is compared against the value of “strAnchor”. When a match is found, the entire entry is extracted from the FAQ using the DOM’s cloneNode method and appended to the copy of Results.xml that is stored in memory. The XSLTemplate processor object transforms the completed document using instructions supplied in the XSL style sheet object. The transformed document is then displayed in the main display frame using the DHTML innerHTML property.

The MatchInput function was definitely the most challenging bit of scripting that I completed in the course of the project. Identifying the DOM objects and methods that I needed in order to create and display the new XML document containing the search results took a great deal of research and trial and error. I also struggled with the design of an effective term matching algorithm. Given my limited programming skills and
experience, and the fact that I was working with a relatively small amount of data, I opted
to focus on optimizing the keywords associated with each FAQ entry, rather than
attempting to program a sophisticated full-text search algorithm. I assigned carefully
tailored keywords to each FAQ entry, using word stems to represent words with variable
suffixes. For example, I use the word stem “reserv” to represent the words reserve,
reserves, reservation, and reservations in the keywords element. I found that the
combination of using tightly controlled keyword vocabulary and ranking based on the
number of terms matched compensated for the simplicity of my matching algorithm,
producing reasonably precise results in my test searches. Whether this approach will
yield satisfactory results under more rigorous testing remains to be seen.

To experiment with Peedy’s capabilities as an online tutor, I developed a script
that prompts Peedy to deliver instruction in completing keyword searches in the library’s
online catalog, Wolf (see AVStutorial.vbs in Appendix C). This was a timely topic
because the library recently migrated to new software that provides enhanced keyword
searching capabilities. In order to take advantage of the new functionality, users need to
understand a few things about the search syntax expected by the new software. The goal
of Peedy’s tutorial is to teach users the basics by guiding them through hands-on
searching exercises in a live catalog session. The Wolf Keyword Searching tutorial page
hosts the tutorial script.

The user accesses the tutorial page via a link on the main help index page (see
Figure 5). The tutorial script starts automatically when the tutorial page loads. The first
ing the script does is maximize the window containing the help frameset. This provides
a consistent frame of reference for setting Peedy’s position on the screen as the user
works through the tutorial and increases the likelihood that the help window will remain visible to the user throughout the tutorial session. To give the user access to specialized controls needed during the tutorial session, two new commands are added to Peedy’s popup menu, “Exit tutorial”, and “Repeat last segment”. When the user exits the tutorial page, the script removes these controls. Finally, if the user previously dismissed Peedy using the ‘Hide’ command, he will automatically reappear when the tutorial page is loaded.

Figure 5: WOLF keyword searching tutorial page

The tutorial is broken down into topical segments to allow the user more control and some flexibility. The web page provides a brief description of the content of each segment. Each Begin button on the page is associated with a different subroutine in the
tutorial script. Clicking on one of these buttons launches the associated subroutine. Each subroutine begins by displaying the Wolf keyword search page in an external window, opening a new window if necessary. Peedy appears to move into this window and begins delivering a specialized sequence of dialogue and animations as he guides the user through the tutorial. Peedy pauses periodically, offering the user the opportunity to practice or explore on her own. He instructs the user to click on him when she is ready to continue with the tutorial. When the user completes a segment, Peedy offers congratulations and provides instructions about how to proceed.

The fact that the script cannot actually see what is happening in the Wolf search window is the biggest obstacle that I encountered in developing this tutorial. I attempted to use the DOM to access properties and events in the external window, but because the script runs in the help website window, rather than the Wolf window, I was denied access consistently. Despite considerable effort, I was unable to find a solution to this problem. As a workaround, I carefully sequenced Peedy’s dialogue and animations to provide pauses in critical places. I also broke the tutorial segments down into even smaller segments and reconfigured Peedy’s left-click behavior so that when the user clicks on him during a tutorial segment, he advances to the next segment, rather than displaying a dialog box to accept user input. Since Peedy has not yet been programmed to respond to questions that a user might have within the context of the tutorial, it was a good idea to suppress Peedy’s default left-click behavior during the tutorial anyway. Though I was able to produce functional tutorials, this functionality is quite tenuous and does not demonstrate Peedy’s full potential as an online tutor. I plan to continue searching for a way to monitor events in external windows since having this capability would allow me
to program Peedy to respond to events that occur in the user’s environment at a much more detailed level.

**Evaluation**

To get some preliminary feedback on the prototype application, I invited Hamersly Library staff and student workers to preview the site and complete an evaluation worksheet (see Appendix D). While this was hardly a rigorous and unbiased evaluation process, it provided some useful suggestions for refinement and further development of the prototype. Of 40 worksheets distributed, 20 were returned within the 10-day evaluation period. All worksheet responses were anonymous, though some individuals also chose to offer some verbal feedback, some of which I have included in this summary.

The first section of the worksheet asked respondents to evaluate the help website home page and the browsable FAQ pages. All of the respondents were satisfied with the response time on the initial page, though it is important to note that all completed the evaluation worksheet from workstations in the Hamersly Library, which have Ethernet connections to the Internet. In my own evaluation and testing, I noticed that it took about twice as long for Peedy to load over a modem connection as it does over an Ethernet connection. Once Peedy’s character file and animations have been loaded, Peedy’s response time is not affected much by the speed of the user’s Internet connection, since he is controlled by a client-side script. Because the pages in the help website frameset are very short and use few images, they always load relatively quickly. Users can start using the site even before Peedy appears, so as long as the load time differential is factored into Peedy’s design carefully, it should not cause major problems. While most
respondents were also satisfied with the display and navigation features offered for the FAQ, several offered suggestions for improving the visual display of the FAQ pages, such as providing more structure and visual definition for the internal links listed at the top of each page and increasing the font size of the question headings and answer text.

Next, respondents were instructed to work through the About Peedy tutorial page. Most were satisfied with the clarity and completeness of the instructions provided and with Peedy’s responsiveness. Several respondents noted that Peedy was unable to answer their questions satisfactorily. In two cases, this was because the FAQ did not contain answers to the questions they asked. In another case, the respondent noted that when asking Peedy a personal question, he suggested visiting the information desk, which is currently his default response. These comments point out the value of supporting Peedy with a rich knowledge base and natural language processing capabilities. Respondents liked Peedy’s personality and they seemed willing to interact with him. They expressed disappointment when his responses didn’t live up to their expectations.

To my surprise, there were few complaints about the Wolf keyword searching tutorial. Few respondents even seemed to notice that Peedy’s actions were not directly related to events that occurred in the Wolf search window. Several who noted that Peedy got out of synch with them or seemed a little slow to respond still expressed satisfaction with the tutorial, saying that the lack of synchronization did not prevent them from reading and learning from Peedy’s instructions. Some respondents indicated that they had difficulty reading and comprehending all of the information displayed in Peedy’s word balloon. A few wished that Peedy’s word balloon would not disappear so quickly after he finished a particular phrase. Unfortunately, MS Agent gives neither the
programmer nor the user any control over this aspect of display. Several asked for the ability to make Peedy repeat instructions or speak more slowly. Apparently, despite having completed the About Peedy tutorial, they were not aware that Peedy already supports this functionality, so I may need to explore options for making these controls more obvious and accessible to users. It is also important to note here that the respondents completed the evaluation using workstations that do not currently support MS Agent’s audio output features, so their comments about Peedy are based solely on silent interactions with Peedy, i.e. reading the spoken text as it appeared in his word balloon. In designing the tutorial, I noticed that this was one area where having both the word balloon and audio output made it significantly easier to comprehend Peedy’s dialogue. One respondent suggested proofing the user’s directed searches for errors, so that Peedy could respond appropriately, rather than simply continuing on with his scripted dialogue as if the search were successful, even when it’s not. This comment underscores the need to have the ability to monitor events in the user’s environment and prompt Peedy to respond when appropriate.

Overall, the response to the prototype website was positive. Most respondents indicated that they had enjoyed using the prototype website and agreed that the FAQ and Peedy, or a character like him, should be incorporated into the library’s regular website. Most respondents agreed strongly that they felt like they had sufficient control over Peedy. One individual made a point of telling me that even though he personally dislikes and never uses Office Assistants in Microsoft products, he felt that Peedy had the potential to provide good support to users of the library website, especially those who are intimidated by technology. Though much more rigorous testing is required before
deciding whether it is worthwhile to develop this prototype into a full-blown web application, the fact that many library staff and student workers responded favorably toward the prototype is an encouraging sign.
Conclusions

Overall, this was a successful project. I achieved my fundamental goal by demonstrating that it is possible to use broadly recognized web standards and freely available software tools to develop an interface agent application for a library website. The prototype website developed in this project provides only a glimpse at the services that an interface agent application could provide to library users, however. In the process of completing this project, I improved my web programming skills and developed a better awareness of the challenges associated with designing and programming an interface agent application. I feel that I am now better prepared to assess the capabilities of software applications designed to support the development of interface agents or to contribute to the development of such applications, should the opportunity arise.

The prototype application could benefit from the inclusion of several additional features. Currently, its ability to monitor the state and events occurring in the user’s session is extremely limited. In addition to figuring out how to monitor events in windows external to the help website, I would like to experiment with using cookies to log user and session state information. This would allow me to save information used to personalize the interface between sessions. I would also like to develop a mechanism for compiling a transaction log of user activity. This would provide valuable information to library staff that could be used to refine and expand the content and features of the system. Implementing a transaction log feature will most likely require some server-side programming.

MS Agent provides the essential building blocks needed to program an interface agent character. It enables individuals with modest programming skills to develop
interface agents, but it provides no support for imbuing the agents with intelligence. It is necessary to combine the services provided by MS Agent with those provided by other systems in order to achieve the functionality I envision for an interface agent. Theoretically, it should be possible to use MS Agent in combination with other systems, such as a knowledge base or expert system engine, but building such infrastructure applications is far beyond my own programming abilities. I would like to experiment with building a more comprehensive system that combines the services of MS Agent with those provided by an expert systems engine software application.

MS Agent’s biggest drawback is that it is totally dependent upon Microsoft-centric technology. This makes it virtually impossible to use it to develop a browser and platform neutral interface agent. This is a major barrier to using MS Agent in building library applications. Philosophically, libraries prefer not to dictate to their users which type of computer or which software they must use in order to access online library services. Where technological barriers do exist, libraries typically try to offer alternatives to ensure that all users receive an equitable level of service. As a librarian, I am torn by my desire to offer innovative services that have the potential to be of real benefit to users and the need to make services as accessible as possible. This is not a dilemma unique to libraries, and this is probably one of the reasons why MS Agent has not been more widely used.

I would like to continue to develop the FAQ as an XML application. To support this development, it would be best to either identify an existing XML editing application to support maintenance of FAQ content or to build a customized application for this purpose. Using Excel for data entry worked reasonably well for the purposes of this
project, but it is not an ideal long-term solution. Even if the size of the FAQ someday grows to the point that migrating the data to a database becomes necessary, the XML framework will still be very useful for presenting and supporting client-side manipulation of the data. I would like to continue searching for a means of providing inline links in the FAQ entries and to enhance the XSL style sheet to use HTML table elements to structure the display of the FAQ entries. This would provide more detailed control of page layout and would allow me to enhance the visual display of the data.

The application needs a more sophisticated search engine for the FAQ. The current system relies heavily upon having carefully tailored keywords assigned to each entry. It might be difficult to sustain this practice in the future, particularly in an environment where multiple individuals are regularly contributing to maintenance of the FAQ. It is also unclear whether this practice would scale well as the amount of data grows. It might be possible to provide some automated support for generating keywords into a FAQ data entry or content management application, but programming this feature might be more complicated than developing or simply acquiring a more sophisticated full-text search algorithm. Users have come to expect sophisticated features like relevancy ranking and automatic detection of spelling errors when they search web pages. The current application does not support these features.

Though preliminary, the results of the evaluation of the prototype website indicate that the interface agent had some of the desired effects. Most respondents liked Peedy and found him to be a potentially positive addition to the library’s website. I hope that with some further development, the prototype can eventually be tested with end-users to
see if using an interface agent on a library website can help to bring some of the sociality found in the physical library to the online environment.
REFERENCES
References


helpindex.htm

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN" "http://www.w3.org/TR/html4/frameset.dtd">
<html>
<head>
<title>Hamersly Library Help</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>

<frameset frameborder="0" framespacing="0" border="0"
rows="10%,70%,20%">
  <frame src="Help/HLbanner.htm" name="banner"
frameborder="no" scrolling="no" noresize>
    <frameset cols="75%,25%" framespacing="0"
frameborder="no">
      <frame src="Help/index.htm" name="Index">
        <frame src="Help/Peedy.htm" name="Peedy"
scrolling="NO" noresize>
        </frameset>
    </frame>
  </frameset>
<noframes><body><h2>This page requires a frames-enabled browser. Return to the <a href="http://www.wou.edu/library/">Hamersly Library homepage</a></h2></body></noframes>

</html>
<title>Hamersly Library Help</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<link rel="stylesheet" href="help.css" type="text/css">
</head>

<body>
<p class="title">Hamersly Library Help</p>
<table>
<tr><th>FAQ</th><th>Tutorials</th></tr>
<tr><td width="50%"><a href="Ref.xml">Reference</a></td><td width="50%"><a href="AVStutorial.htm">WOLF: Keyword Searching</a></td></tr>
<tr><td width="50%"><a href="Circ.xml">Borrowing &amp; Reserves</a></td><td width="50%"></td></tr>
<tr><td width="50%"><a href="Ill.xml">Interlibrary Loan</a></td><td width="50%"></td></tr>
<tr><td width="50%"><a href="Vocab.xml">Glossary</a></td><td width="50%"></td></tr>
<tr><td width="50%"><a href="FAQ.xml">Entire FAQ</a></td><td width="50%"></td></tr>
</table>
<p>Last modified:<br/>
<!-- #BeginDate format:fcAmla -->Monday, April 7, 2003 7:46 PM<!-- #EndDate -->
</p>
</body>
</html>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Peedy's Page</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<OBJECT ID="Agent" width=0 height=0
CLASSID="CLSID:D45FD31B-5C6E-11D1-9EC1-00C04FD7081F"
CODEBASE="#VERSION=2,0,0,0">
</OBJECT>
<script language="VBScript" src="PeedyHelp.vbs" type="text/VBScript"></script>
</head>
<body bgcolor="#FCFFD5">
  <input type=text id="text1" name="text1"><br>
  <input type=button id="FindButton" value="Ask" onclick="Analyze(text1.value)">
  <input type=button id="SpeakButton" value="Speak" onclick="Speak(text1.value)">
  <p></p>
  <input type=button id="CallButton" value="Call Peedy" onclick="CallPeedy()">
  <input type=button id="JokeButton" value="Tell a Joke" onclick="TellJoke()">
  <input type=button id="BioButton" value="About Peedy" onclick="AboutPeedy()">
</body>
</html>
About Peedy

This page introduces Peedy, an animated interface agent that has been programmed to interact with users of the Hamersly Library website. Peedy was developed using Microsoft Agent and VB Script to program me.

To learn more about this site, click the 'About site' button.

End Sub

53
Microsoft Agent, a software development kit (SDK) freely available from Microsoft. The code that controls Peedy was written entirely in VBScript. To learn more about how this site is structured, click on the 'About site' button below.

The instructions below describe how to control Peedy.

- **Move Peedy:**
  - Sometimes, Peedy gets in your way. If this happens, you can move him by Left-clicking on him and dragging him somewhere else. He really doesn't mind being moved, but sometimes he'll have something to say about it. Go ahead and move him a couple of times to see how he responds.

- **Hide Peedy:**
  - Sometimes, you just want Peedy to go away. To do this, just Right-click on Peedy and select 'Hide'. If you want to call Peedy back after you've hidden him, just click on the 'Call Peedy' button on Peedy's page.

- **Ask Peedy to repeat himself:**
  - Sometimes, Peedy talks too darned fast and you miss what he said. To make Peedy repeat the last thing he said, just Right-click on him and select 'Repeat last statement' from the menu.

- **Stop Peedy:**
  - Sometimes, you simply want Peedy to stop whatever it is he's doing. To make Peedy stop, simply Right-click on him and select 'Stop' from the menu.
<dt>Advanced Character Options</dt><br>
<input type=button value="Narrate"
onclick="top.Peedy.AboutOptions()"
><dd>To access Peedy's Advanced Character Options menu, simply Right-click on him and select 'Advanced Character Options'. You can adjust the speed of Peedy's speech by adjusting the control under the 'Output' tab of Peedy's Advanced Character Options menu.</dd><br>
<dt>Teach Peedy to Speak</dt><br>
<input type=button value="Narrate"
onclick="top.Peedy.AboutSpeak()"
><dd>Peedy is a parrot, so he likes to repeat things that people say. To make Peedy speak, just enter the phrase that you'd like him to repeat into the box on Peedy's page and click the Speak button. Unfortunately, Shirley hasn't yet programmed Peedy to remember the words that you teach him. Maybe someday she'll have time to add that capability.</dd><br>
<dt>Talk to Peedy</dt><br>
<input type=button value="Narrate"
onclick="top.Peedy.AboutTalk()"
><dd>Normally when you Left-click on Peedy, a dialog box will appear. Just type the question you'd like to ask into the box and click OK (or press the enter key) to see how Peedy responds. Entering a question into the box on Peedy's page and clicking the Ask button will do the same thing. For example:
<ol>
<li>Left-click on Peedy, enter: 'How are you today, Peedy' into the dialog box, and click OK.</li>
<li>Click on the 'Tell a joke' button</li>
<li>Use the box on Peedy's page and the 'Ask' button to ask Peedy how to reserve a group study room</li>
</ol></dd>
</dt>
<p align="center"><a href="index.htm">Back to Help Index</a></p>
AVStutorial.htm

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Hamersly Library Help: WOLF Keyword Searching Tutorial</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<link rel="stylesheet" href="help.css" type="text/css">
</head>
<BODY>
<script language="VBScript" src="AVStutorial.vbs" type="text/vbscript"></script>
<p class="title"> WOLF Keyword Searching Tutorial </p>
<p align="center"><a href="index.htm">Back to Help Index</a></p>
<p class="italics">This tutorial provides an introduction to the keyword searching features available in the <a href="http://library.wou.edu/" target="Display">WOLF online catalog</a></p>
<dl>
<dt>Part 1: Phrase searching</dt>
<dd class="desc">Get acquainted with the Wolf keyword searching interface and learn to do basic Phrase searching.</dd>
<table cellspacing="20">
<td><input type=button value="Begin" onclick="Part1()"></td>
<td class="times">To exit the tutorial at any time, Right-click on Peedy and select 'Exit tutorial'</td>
</table>
<dt>Part 2: Boolean searching</dt>
<dd class="desc">Learn how to use Boolean operators 'And', 'Or', & 'And Not' in Wolf keyword searches</dd>
<table cellspacing="20">
<td><input type=button value="Begin" onclick="Part2()"></td>
<td class="times">To exit the tutorial at any time, Right-click on Peedy and select 'Exit tutorial'</td>
</table>
<dt>Part 3: Using truncation & wildcard symbols</dt>
<dd class="desc">Learn how to use truncation and wildcard symbols to expand your searches</dd>
</dl>
To exit the tutorial at any time, Right-click on Peedy and select 'Exit tutorial'.

Part 4: Field searching
Coming soon

Part 5: Proximity searching
Coming soon

Back to Help Index.
FAQtoXML.vbs

' FAQ to XML script
' Created: 1-27-03
' Last updated: 4-6-03
' This script converts a tab-delimited text file, output
' from MS Excel, to an XML
' document

'Define string constants that will be used later
Const TITLE = "FAQ to XML"
DOCTAG = Chr(60) & "document" & Chr(62)
XDOCTAG = Chr(60) & Chr(47) & "document" & Chr(62)
ENTRYTAG = Chr(9) & Chr(60) & "entry" & Chr(62)
XENTRYTAG = Chr(9) & Chr(47) & "entry" & Chr(62)
QUESTIONTAG = Chr(9) & Chr(9) & Chr(60) & "question" & Chr(62)
XQUESTIONTAG = Chr(60) & Chr(47) & "question" & Chr(62) & Chr(10)
ANSWERTAG = Chr(9) & Chr(9) & Chr(60) & "answer" & Chr(62)
XANSWERTAG = Chr(60) & Chr(47) & "answer" & Chr(62) & Chr(10)
CATTAG = Chr(9) & Chr(9) & Chr(60) & "category" & Chr(62)
XCATTAG = Chr(60) & Chr(47) & "category" & Chr(62) & Chr(10)
ANCHORTAG = Chr(9) & Chr(9) & Chr(60) & "anchor" & Chr(62)
XANCHORTAG = Chr(60) & Chr(47) & "anchor" & Chr(62) & Chr(10)
KEYWORDSTAG = Chr(9) & Chr(9) & Chr(60) & "keywords" & Chr(62)
XKEYWORDSTAG = Chr(60) & Chr(47) & "keywords" & Chr(62) & Chr(10)
SEEALSOTAG = Chr(9) & Chr(9) & Chr(60) & "seealso" & Chr(62)
XSEEALSOTAG = Chr(60) & Chr(47) & "seealso" & Chr(62) & Chr(10)
TUTORIALTAG = Chr(9) & Chr(9) & Chr(60) & "tutorial" & Chr(62)
XTUTORIALTAG = Chr(60) & Chr(47) & "tutorial" & Chr(62) & Chr(10)

'Declare variables used in the script
Dim outputFSO, inputFSO, OrigFileName, NewFileName, FAQFSO
Dim TXTfile, XMLfile, NextLine, Tab, text, quotePos,
dblquotePos, cleantext, FAQfile
Dim q, a, category, anchor, seealso, tutorial
Dim intAgain

intAgain = 6

'Get the name of the file to convert
OrigFileName = GetFileName

'Create name of XML file based on text file name
NewFileName = Mid(OrigFileName, 1, Len(OrigFileName) - 4) & ".xml"
MsgBox NewFileName, vbOKOnly,TITLE

'Create output files
Set outputFSO = CreateObject("Scripting.FileSystemObject")
Set faqFSO = CreateObject("Scripting.FileSystemObject")
Set XMLfile = outputFSO.CreateTextFile(NewFileName)
Set FAQfile = faqFSO.CreateTextFile("o:\FAQ\FAQ.xml")

Start(XMLfile)
Start(FAQfile)

'Open and begin reading from text file
Set inputFSO = CreateObject("Scripting.FileSystemObject")
Set TXTfile = inputFSO.OpenTextFile(OrigFileName)

Do While intAgain = 6
NewLine = TXTfile.ReadLine 'read in top line of file, containing field labels

' Parse data from the text file and construct the XML file

Do Until TXTfile.AtEndOfStream
    NewLine = TXTfile.ReadLine
    Tab = InStr(NewLine,vbTab)
    q = Left(NewLine,Tab - 1)
    q = TrimQuote(q)
    If q <> "" Then
        XMLfile.WriteLine(ENTRYTAG)
        FAQfile.WriteLine(ENTRYTAG)
        XMLfile.Write (QUESTIONTAG & Trim(q) & 
                        XQUESTIONTAG)
        FAQfile.Write (QUESTIONTAG & Trim(q) & 
                        XQUESTIONTAG)
        NewLine = Mid(NewLine,Tab + 1)
        Tab = InStr(NewLine,vbTab)
        a = Left(NewLine,Tab - 1)
a = TrimQuote(a)
XMLfile.Write (ANSWERTAG & Trim(a) & XANSWERTAG)
FAQfile.Write (ANSWERTAG & Trim(a) & XANSWERTAG)
NewLine = Mid(NewLine,Tab + 1)
Tab = InStr(NewLine,vbTab)
category = Left(NewLine,Tab - 1)
XMLfile.Write (CATTAG & Trim(category) & XCATTAG)
FAQfile.Write (CATTAG & Trim(category) & XCATTAG)
NewLine = Mid(NewLine,Tab + 1)
Tab = InStr(NewLine,vbTab)
anchor = Left(NewLine,Tab - 1)
anchor = TrimQuote(anchor)
XMLfile.Write (ANCHORTAG & Trim(anchor) & XANCHORTAG)
FAQfile.Write (ANCHORTAG & Trim(anchor) & XANCHORTAG)
NewLine = Mid(NewLine,Tab + 1)
Tab = InStr(NewLine,vbTab)
keywords = Left(NewLine,Tab - 1)
keywords = TrimQuote(keywords)
XMLfile.Write (KEYWORDSTAG & Trim(keywords) & XKEYWORDSTAG)
FAQfile.Write (KEYWORDSTAG & Trim(keywords) & XKEYWORDSTAG)
NewLine = Mid(NewLine,Tab + 1)
Tab = InStr(NewLine,vbTab)
seealso = Left(NewLine,Tab - 1)
seealso = TrimQuote(seealso)
XMLfile.Write (SEEALSOTAG & Trim(seealso) & XSEEALSOTAG)
FAQfile.Write (SEEALSOTAG & Trim(seealso) & XSEEALSOTAG)
NewLine = Mid(NewLine,Tab + 1)
tutorial = NewLine
tutorial = TrimQuote(tutorial)
XMLfile.Write (TUTORIALTAG & Trim(tutorial) & XTUTORIALTAG)
FAQfile.Write (TUTORIALTAG & Trim(tutorial) & XTUTORIALTAG)
XMLfile.WriteLine(XENTRYTAG)
FAQfile.WriteLine(XENTRYTAG)
Else
Exit Do
End If
Loop
TXTfile.Close
XMLfile.WriteLine(XDOCTAG)
XMLfile.Close

intAgain = MsgBox (NewFileName & " created. Process another file?", 4, TITLE)

If intAgain = 7 Then
    FAQfile.WriteLine(XDOCTAG)
    FAQfile.Close
    Exit Do
Else
    OrigFileName = GetFileName
    NewFileName = Mid(OrigFileName, 1, Len(OrigFileName) - 4) & ".xml"
    sheetName = Mid(OrigFileName, 4, Len(NewFileName) - 7) & ".xsl"
    MsgBox NewFileName,vbOKOnly,TITLE
    Set XMLfile = outputFSO.CreateTextFile(NewFileName)
    Set TXTfile = inputFSO.OpenTextFile(OrigFileName)
    Start(XMLfile)
End if
Loop

MsgBox "Thank you for using FAQtoXML.vbs", 0, TITLE

Function GetFileName()
    GetFileName = InputBox("Please enter the file path and name of the file that you wish to convert to XML.", TITLE)
End Function

Sub Start(outFile)
    outFile.WriteLine (Chr(60) & Chr(63) & "xml version" & Chr(61) & Chr(34) & "1.0" & Chr(34) & " encoding" & Chr(61) & Chr(34) & " iso-8859-1" & Chr(34) & Chr(63) & Chr(62))
    outFile.WriteLine (Chr(60) & Chr(63) & "xml-stylesheet type" & Chr(61) & Chr(34) & "text/xsl" & Chr(34) & " href=" & Chr(34) & "help.xsl" & Chr(34) & Chr(63) & Chr(62))
    outFile.WriteLine(DOCTAG)
End Sub

'Trims leading and trailing double quote characters from a string
Function TrimQuote(ByVal text)
    text = Trim(text)
    quotePos = 0
    quotePos = InStr(text,Chr(34))
End Function
If quotePos = 1 Then
    text = Mid(text,2,Len(text) - 2)
End If

text = DblQuote(text)
TrimQuote = text

'Removes one set of double quote characters from a string
when 2 sets are encountered
'adjacent to each other
Function DblQuote(ByVal text)
    text = Trim(text)
    dblquotePos = 0
    dblquotePos = InStr(text,Chr(34) & Chr(34))
    cleantext = ""

    Do While dblquotePos > 0
        cleantext = cleantext & Mid(text,1,dblquotePos)
        text = Mid(text, dblquotePos + 2, Len(text))
        dblquotePos = InStr(text, Chr(34) & Chr(34))
    Loop

    cleantext = cleantext & text

    DblQuote = cleantext
End Function
<xsl:stylesheet version="1.0" encoding="iso-8859-1" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <head>
        <title>Hamersly Library FAQ</title>
        <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
        <link rel="stylesheet" href="help.css" type="text/css" />
      </head>
      <body>
        <p class="title"><a name="top"> FAQ </a></p>
        <p align="center"><a href="index.htm">Back to Help index</a></p>
        <xsl:apply-templates select="/document/entry" mode="Index">
          <xsl:sort select="anchor" />
        </xsl:apply-templates>
        <hr />
        <xsl:for-each select="/document/entry">
          <xsl:sort select="anchor" />
          <p class="question"><a><xsl:attribute name="name"><xsl:value-of select="anchor" /></xsl:attribute><xsl:value-of select="question" /></a><img src="Image2.jpg" /><a href="#top">Back to Top</a></p>
          <p class="answerVis"><xsl:value-of select="answer" /></p>
          <p><xsl:apply-templates select="answer/a" /></p>
        </xsl:for-each>
        <hr />
        <p align="center"><a href="index.htm">Back to Help index</a></p>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="/document/entry" mode="Index">
    <a><xsl:attribute name="href="#"><xsl:value-of select="anchor" /></xsl:attribute><xsl:value-of select="question" /></a><br />
  </xsl:template>
  <xsl:template match="a">
    64
  </xsl:template>
</xsl:stylesheet>
<document>
  <entry>
    <question>How long can I keep something that I have checked out?</question>
    <answer>WOU students and community borrowers can check out books for 3 weeks. WOU faculty and staff can check out books for 6 months.</answer>
    <category>General</category>
    <anchor>loanper</anchor>
    <keywords>loan period borrow how long check keep</keywords>
    <seealso></seealso>
    <tutorial></tutorial>
  </entry>
  <entry>
    <question>Can I renew materials that I have checked out?</question>
    <answer>Patrons can renew materials checked out from Hamersly Library or Orbis once through the <a href="https://library.wou.edu/patroninfo">View your Library Record</a> option in WOLF. If you need to renew materials more than once, or if you need additional assistance, please call library's checkout desk at (503) 838-8902.</answer>
    <category>General</category>
    <anchor>renewpol</anchor>
    <keywords>renew check allow how book online</keywords>
    <seealso></seealso>
    <tutorial></tutorial>
  </entry>
  <entry>
    <question>How many books can I check out at one time?</question>
    <answer>Patrons may have up to 300 items checked out at any given time.</answer>
    <category>Books</category>
    <anchor>checklim</anchor>
    <keywords>maximum check limit book how many borrow</keywords>
    <seealso></seealso>
    <tutorial></tutorial>
  </entry>
</document>
<entry>
    <question>Can I check out magazines?</question>
    <answer>WOU students can check out magazines for one day. WOU faculty and staff can check out magazines for one week. Community borrowers are not allowed to check out magazines, but copies of articles can be made using photocopiers available in the copy center on the first floor.</answer>
    <category>Journals</category>
    <anchor>percheck</anchor>
    <keywords>periodical journal magazine borrow loan lend check</keywords>
</entry>

<entry>
    <question>How much does it cost to make a photocopy?</question>
    <answer>10 cents per page when paying in cash and 5 cents per page with a copy card. Copy cards can be purchased for $1.00 from a vending machine in the first floor copy center. Once purchased, the card can be "recharged" with monetary value indefinitely.</answer>
    <category>Photocopies</category>
    <anchor>copycost</anchor>
    <keywords>photocop cost price charge how much</keywords>
</entry>

<entry>
    <question>How do I get something that's on e-reserve?</question>
    <answer>Visit our <a href="http://www.wou.edu/library/reserves/index.htm">Reserves page</a> and search by course number or by your professor's last name. When an item is on e-reserve, you will see a View link. Just click on that link to view the full-text. See our <a href="http://www.wou.edu/provost/library/reserves/usingeres.htm">Using e-reserves page</a> for more information about e-reserves.</answer>
    <category>Reserves</category>
    <anchor>reselec</anchor>
    <keywords>e-reserve ereserve electronic reserve course class professor reading</keywords>
</entry>
<question>Where can I find something that's on reserve?</question>

<answer>Search <a href="http://www.wou.edu/provost/library/reserves/index.htm">WOLF reserves</a> by course number or your professor's last name to retrieve the list of materials on reserve for your class. If the listing says "Electronic Resources," click on View to access the materials. If the listing says "Checkout Desk," write down the call number and take it to the checkout desk, where a library staff member will retrieve the item for you.</answer>

<category>Reserves</category>
<anchor>resfind</anchor>
<keywords>reserve course textbook reading class find</keywords>
<br/><seealso></seealso>
<br/><tutorial></tutorial>

<question>My professor told me that he had left something in the library for everyone in my class to read. Where is it?</question>

<answer>When professors leave something at the library for their students, we generally place it on reserve. To locate the list of materials that are on reserve for a specific professor or course, search <a href="http://www.wou.edu/provost/library/reserves/index.htm">WOLF reserves</a> by course number or your professor's last name. If you have difficulty locating an item, please ask for help at the Information desk or call (503) 838-8418.</answer>

<category>Reserves</category>
<anchor>resdef</anchor>
<keywords>reserve course class textbook read professor left</keywords>
<br/><seealso></seealso>
<br/><tutorial></tutorial>

<question>How do I place materials on reserve in the library?</question>

<answer>Our <a href="http://www.wou.edu/provost/library/reserves/faq.htm">68</a>
How to place materials on reserve page provides full details on placing materials on reserve in the library.

<category>Reserves</category>
<anchor>resfac</anchor>
<keywords>reserve plac how class course faculty</keywords>

What supplies does the Library sell?

The library has a limited number of supplies available for purchase at the Checkout desk. We sell floppy disks, blank transparency film, CD-R and CD-RW discs, and Copy cards.

Can I view my personal record in your system?

Go to the Wolf View your Library Record page and enter your full name and the 'V-number' from your student ID. From this page, you can view a list of all of the books that you currently have checked out and view the status of any outstanding Orbis requests. You can also update your mailing address or input your preferred e-mail address.

Wolf and Orbis keep asking me to enter my 'V-number'. What is a V-number and where can I find out what mine is?

WOU no longer uses Social Security numbers as campus ID numbers. Now, all WOU faculty, staff
and students have been assigned a unique ID number that starts with a 'V', hence the name 'V-number'. Most members of the campus community have been issued a new ID card in the past year that lists their V-number. You can look up your V-number on the University's <a href="http://wilbur.wou.edu/pls/wou/wou_web.vendor_info.get_info">SSN to Vendor number page. </a></answer>  
  
  <category>General</category>  
  <anchor>idnum</anchor>  
  <keywords>V-number ID number WOU student vendor social security SSN</keywords>  
  <seealso></seealso>  
  <tutorial></tutorial>  

</entry>  
</document>
' Created: 3/4/03
' Last updated: 4/30/03
' Code to control an MS Agent character for the Hamersly Library Help website

'************************************
'Global variables
'************************************
'Object variable for Agent character
Dim Peedy

'Request object variables
Dim LoadRequest, GetShowAnimation

'String variable to for the user's name
Dim UsrName

'Integer variables
Dim greet, drag, click, part, stopDemo

'String variables
Dim answer, strBody, prompt

'Object variable for FAQ results display
Dim objResults

'************************************
'Initialization routines
'************************************
Sub Window_OnLoad()
    greet = 0
    drag = 1
    click = 1
    part = 0
    stopDemo = 0
    UsrName = ""
    quote = ""
    answer = ""
    prompt = "Please enter a question for Peedy."
    LoadCharacter
End Sub

Sub LoadCharacter()
    On Error Resume Next

Agent.Connected = True   ' necessary for IE3

'Use this line to load Peedy from the local hard drive:
'   Set LoadRequest = Agent.Characters.Load ("Peedy", "peedy.acs")

'Use this line to load Peedy from the Microsoft website:

   window.status = "Peedy loading"
End Sub

Sub LoadAnimation()
   Set Peedy = Agent.Characters("Peedy")
   Set GetShowAnimation = Peedy.Get ("state", "showing, speaking, hiding, moving")
   Peedy.Get "animation", "Blink, Greet, GreetReturn", False
   Peedy.Get "state", "gesturing, idling"
   Peedy.Get "animation", "Alert, Acknowledge, Confused, Decline, Don't Recognize, Get Attention, Pleased, Explain, Think, Idle1_1, Idle1_2, Idle2_1, Idle2_2, Uncertain, Rest Pose, Reading, Surprised, Wave", False
End Sub

'Set default commands available in Peedy's popup menu
Sub InitCommands()
   Peedy.Commands.RemoveAll
   Peedy.Commands.Add "Stop", "Stop"
   Peedy.Commands.Add "Repeat", "Repeat last statement"
   Peedy.Commands.Add "ACO", "Advanced Character Options"
End Sub

'****************************
'MS Agent event handlers
'****************************
Sub Agent_RequestComplete(ByVal Request)
   If Request = LoadRequest Then
      window.status = "Peedy loaded, animations loading."
      LoadAnimation
   End If
End Sub
If Request = GetShowAnimation Then
    window.status = "Peedy animations loaded."
    DoIntro
End If
End Sub

Sub Agent_Click (ByVal CharacterID, ByVal Button, ByVal Shift, ByVal X, ByVal Y)
    If Button <> 2 Then
        Select Case click
            Case 0
                Peedy.Stop
                Peedy.GestureAt window.screenLeft, window.screenTop+300
                Peedy.Speak "Please enter your first name in the box."
                UsrName = GetName
                While UsrName = "doagain"
                    Peedy.GestureAt window.screenLeft, window.screenTop+200
                    Peedy.Speak "Please enter only the name you'd like me to use in the box."
                    UsrName = GetName
                Wend
                Peedy.Play "Wave"
                Peedy.Speak "Hi " & UsrName & " it's nice to meet you. Just click on me if you want to ask a question. "
                prompt = "Please enter a question for Peedy"
                click = 1
            Case 1
                Peedy.Stop
                Peedy.Play "Alert"
                If UsrName = "" Then
                    Peedy.Speak "What's up?"
                Else
                    Peedy.Speak "What's up, " & UsrName & "?"
                End if
                InputWin
            Case 2
                PickPart(part)
        End Select
    End If
End Sub

Sub Agent_DragComplete (ByVal CharacterID, ByVal Button, ByVal Shift, ByVal X, ByVal Y)
If drag = 1 Then
    Peedy.Stop
    Peedy.Play "Blink"
    Peedy.Speak "Well, this gives me a new perspective! I like it much better over here. He, he, he That tickles!"
    Peedy.Play "Blink"
End If

Sub Agent_Command (ByVal UserInput)
    Dim ans
    Select Case UserInput.Name
    Case "ACO"
        Agent.PropertySheet.Visible = True
    Case "Repeat"
        Peedy.Speak "Lst"
    Case "Stop"
        If stopDemo = 1 Then
            Peedy.Stop
            Peedy.Play "GetAttention"
            Peedy.Speak "Good job!"
            stopDemo = 0
        Else
            Peedy.Stop
        End If
    Case "About"
        AboutPeedy
    Case "ExitTut"
        Peedy.Stop
        Peedy.Play "Alert"
        Peedy.Speak "Are you sure you want to exit the tutorial now?"
        ans = MsgBox("Exit tutorial now?", 4, "Talk to Peedy")
        If ans = 6 Then
            Peedy.Play "Wave"
            Peedy.Speak "Ok, I hope to see you back here soon. Bye, Bye."
            click = 1
            drag = 1
            part = 0
            Peedy.Commands.Remove "ExitTut"
            window.open "index.htm", "Index"
        Else
...
Sub DoIntro()
    InitCommands
    Peedy.LanguageID = &H0409  ' needed under some conditions (English)
    On Error Resume Next
    Peedy.MoveTo window.screenLeft, window.screenTop+200
    Peedy.Show
    Peedy.Play "Greet"
    If greet = 0 Then
        Peedy.Speak "Hi, my name is Peedy."
        Peedy.Play "Blink"
        Peedy.Speak "I'm not as smart as a librarian, but I can search for answers to your questions about the Hamersly Library and its services in the Library's Frequently Asked Questions list."
        Peedy.Play "Explain"
        Peedy.Speak "I also know a few jokes, and I can answer a few personal questions about myself. Just click on me if you want to introduce yourself."
        greet = 1
        click = 0
        Peedy.Play "Blink"
        Peedy.Play "Idle2_1"
    End If
End Sub

Function Get_Name()
Dim nameInput
prompt = "Please enter your first name"
nameInput = InputBox(prompt, "Talk to Peedy")
nameInput = NameProcess(nameInput)
GetName = nameInput
End Function

Function NameProcess(UserInput)
Dim words, begName, endName, nameInput, endText
Dim yesorno

UserInput = Trim(UserInput)

If StrComp(Left(UserInput, 10), "My name is", 1) = 0 Then
    UserInput = Mid(UserInput, 11, Len(UserInput))
End If

If StrComp(Left(UserInput, 3), "I'm", 1) = 0 Then
    UserInput = Mid(UserInput, 4, Len(UserInput))
End If

callme = InStr(1, UserInput, "call me", 1)
If callme > 0 Then
    endText = Len(UserInput) - (callme + 6)
    UserInput = Trim(Right(UserInput, endText))
End If

If Lcase(UserInput) = "dick" Then
    UserInput = UserInput
Else
    UserInput = Sanitize(UserInput)
End If

words = Split(UserInput)
If UBound(words) < 2 Then
    If UserInput <> "" Then
        begName = Ucase(Left(Trim(UserInput), 1))
        endName = Lcase(Mid(Trim(UserInput), 2,
            Len(UserInput)))
        UserInput = begName & endName
    NameProcess = UserInput
    Else
        NameProcess = ""
    End if
Else
    Peedy.Play "Surprised"
End If
Peedy.Speak "You seem to have a really long first name! I don't know if I can remember all of it."
Peedy.Play "Think"
Peedy.Speak "Can I just call you " & words(0) & "?"
yesorno = MsgBox ("Please click Yes or No to answer Peedy's question", 4, "Talk to Peedy")
If yesorno = 6 Then
    nameInput = words(0)
    NameProcess = nameInput
Else
    NameProcess = "doagain"
End If
End Function

'* ******************************
'* User input processing routines
'* ******************************
Sub InputWin()
    Dim recognize

    'Display dialog box for accepting user input
    UserInput = InputBox(prompt, "Talk to Peedy")

    'Process user input or respond to lack of input
    If UserInput <> "" Then
        UserInput = Sanitize(UserInput)
        recognize = Analyze(UserInput)
        If recognize = "unknown" Then
            Peedy.Speak "You said: " & UserInput
        End If
    Else
        Peedy.Play "DontRecognize"
        Peedy.Speak "Did you say something?"
        Peedy.Play "Blink"
    End If
    Peedy.Play "Idle1_4"
End Sub

Function Sanitize(text)
    Dim cleanText, arBadwords
cleanText = text

For y = 0 To UBound(arBadwords)
    If InStr(1, text, arBadwords(y), 1) Then
        cleanText = Replace(cleanText, arBadwords(y), " bleep ", 1, -1, 1)
    End If
Next
Sanitize = cleanText
End Function

Function Analyze (phrase)
    Dim quest, resp
    Dim strPhrase, arPhrase

    strPhrase = phrase
    strPhrase = Sanitize(strPhrase)

    'Handle blank input submitted using the Ask button on Peedy's page
    If strPhrase = "" Then
        Peedy.Stop
        Peedy.Speak "Please enter your question in the box and then click the 'Ask' button."
        Analyze = "nothing"
    Else

        quest = 0
        resp = 0

        If InStr(strPhrase, Chr(63)) Then
            quest = 1
        End If

        If InStr(1, strPhrase, "how are you", 1) Then
            resp = 1
        End If

        If InStr(1, strPhrase, "favorite color", 1) Then
            resp = 2
        End If

        If InStr(1, strPhrase, "joke", 1) Then
            resp = 3
        End If
    End If
End Function
If InStr(1, strPhrase, "favorite food", 1) Then
    resp = 4
End If

If InStr(1, strPhrase, "to eat", 1) Then
    resp = 4
End If

If InStr(1, strPhrase, "what's up", 1) Then
    resp = 5
End If

Analyze = Respond(quest, resp, strPhrase)
End If
End Function

Function Respond(quest, resp, strPhrase)
    Dim FAQmatch
    Peedy.Stop
    Select Case resp
        Case 1
            Peedy.Play "Alert"
            Peedy.Speak "I'm fine. Thanks for asking."
            Peedy.Play "Blink"
        Case 2
            Peedy.Play "Alert"
            Peedy.Speak "My favorite color is purple."
            Peedy.Play "Blink"
        Case 3
            TellJoke
        Case 4
            Peedy.Play "Alert"
            Peedy.Speak "Why, \emp\crackers, of course!"
            Peedy.Play "Idle2_2"
        Case 5
            Peedy.Play "Alert"
            Peedy.Speak "Not much. I'm just hanging out."
            Peedy.Play "Idle3_1"
        Case Else
            If quest = 1 Then
                FAQmatch = Find(strPhrase)
                If FAQmatch = "no" Then
                    Peedy.Play "Confused"
                End If
            End If
    End Select
End Function
Peedy.Speak "I'm sorry. I don't know how to answer that question. Please ask at the Information desk or call (503) 838-8418."

Else
    Peedy.GestureAt window.screenLeft-300, window.screenTop+200
    Peedy.Speak "I've found some information in the library's Frequently Asked Questions list that may answer your question."
    Peedy.Play "Blink"
    Peedy.Speak "If you don't find a satisfactory answer here, please ask at the Information desk or call (503) 838-8418."
    End If
Else
    FAQmatch = Find(strPhrase)
    If FAQmatch = "no" Then
        Peedy.Play "Confused"
        Peedy.Speak "I'm sorry. I don't understand what you said."
        Respond = "unknown"
    Else
        Peedy.GestureAt window.screenLeft-300, window.screenTop+200
        Peedy.Speak "I've found some information in the library's Frequently Asked Questions list that may interest you."
        Peedy.Play "Blink"
        Peedy.Speak "If you don't find the information you need here, please ask at the Information desk or call (503) 838-8418."
        End If
    End If
End Select
    Peedy.Play "Idle1_2"
End Function

'**************************
'FAQ searching routines
'**************************
Function Find(text1)
    Dim Found
    If text1 <> "" Then
        Found = MatchInput(text1)
    Else
        Found = "no"
    End If
End Function
Function MatchInput(text1)
    Dim objXmlDoc, Quest, Key, Anchor, newXmlDoc, xslDoc, xslTemplate, proc, newRoot, docRoot
    Dim currNode, MyNewNode, resultDisp
    Dim strBody, strKeywords, strTerms, strAnchor, hit, arKeywords, arTerms
    Dim rank, highrank
    strTerms = Extract(Lcase(text1))
    arTerms = Split(strTerms)
    strBody = ""
    strKeywords = ""
    strAnchor = ""
    rank = 0
    highrank = 1

    Set objXmlDoc = CreateObject("Microsoft.XMLDOM")
    objXmlDoc.async = "false"
    objXmlDoc.Load ("FAQ.xml")
    Set docRoot = objXmlDoc.documentElement
    ' Set docRoot = objXmlDoc.childNodes(2)'this is the
document element
    Set Key =
    objXmlDoc.selectNodes("/document/entry/keywords")
    Set Anchor =
    objXmlDoc.selectNodes("/document/entry/anchor")

    Set newXmlDoc = CreateObject("Msxml2.DOMDocument")
    newXmlDoc.async = "false"
    newXmlDoc.Load ("Results.xml")
    Set newRoot = newXmlDoc.documentElement

    Set xslDoc = CreateObject
    ("MSXML2.FreeThreadedDOMDocument")
    Set xslTemplate = CreateObject
    ("MSXML2.XSLTemplate.3.0")
    xslDoc.async = "false"
    xslDoc.Load ("help.xsl")
    xslTemplate.stylesheet = xslDoc

    Set proc = xslTemplate.createProcessor

    'Check for keywords in entries that match one or more terms
    input by the user
    For i = 1 to Key.length

'Read the terms contained in the <keyword> element for this entry into a string variable & array

strKeywords = Key.item(i-1).text
arKeywords = Split(Lcase(strKeywords))

For s = 0 to UBound(arTerms)
    For x = 0 to UBound(arKeywords)
        If InStr(arTerms(s), Trim(arKeywords(x))) Then
            MsgBox "Term: " & arTerms(s) & Chr(13) & "Keyword: " & arKeywords(x)
            rank = rank + 1
        End If
    Next
Next
If rank > 0 Then
    If rank > highrank Then
        highrank = rank
        strAnchor = Anchor.item(i-1).text & Chr(32)
        MsgBox "High rank: " & highrank
    Else If rank = highrank Then
        hit = Anchor.item(i-1).text
        If InStr(strAnchor, hit) = 0 Then
            strAnchor = strAnchor & Anchor.item(i-1).text & Chr(32)
        End If
    End If
End If
End If
End If
strKeywords = ""
rank = 0
Next

'Extract matching entries from FAQ.xml and write them to a new Xml document
If strAnchor <> "" Then
    For i = 1 to Anchor.length
        If InStr(strAnchor, Anchor.item(i-1).text) Then

            Set currNode = docRoot.childNodes(i-1)
Set MyNewNode = currNode.cloneNode(true)
newRoot.appendChild(MyNewNode)
strBody = "Matching entries extracted"
End If
Next
Else
strBody = ""
End If

'Display search results
If strBody = "" Then
strBody = "<H1>" & strTerms & " not found. </H1>"
render(strBody)
mat = "no"
Else
proc.input = newXmlDoc
proc.transform()
resultDisp = proc.output
Set objResults = window.open("","Index")
objResults.document.body.innerHTML = resultDisp
mat = "yes"
End If
MatchInput = mat
End Function

'Remove stoplisted words from the user input
Function Extract(terms)
Dim stopwatch, arStopList
arStopList = Array(" i ", " a ", " an ", " of ", " for ", " on ", " if ", " the ")
For a = 0 to UBound(arStopList)
stopword = InStr(1, terms, arStopList(a), 1)
If stopwatch > 0 Then
terms = Replace(terms, arStopList(a), " ", 1, -1, 1)
End If
End Function
' MsgBox terms

Extract = terms

End Function

'Display failed search results screen
Sub render(body)
    Set objResults = window.open("","Index")
    objResults.document.write "<html><head><title>Hamersly Library FAQ: Search Results</title><link rel=" & Chr(34) & "stylesheet" & Chr(34) & " href=" & Chr(34) & "help.css" & Chr(34) & "</title><body><H1>FAQ Search Results</H1>" & vbCRLF & "& body & vbCRLF & "<hr><a href=" & Chr(34) & "index.htm" & Chr(34) & ">Back to Help index</a></body></html>"
    objResults.document.close()
End Sub

'***************************************************
'Calls the segment of the AVS tutorial to play based on the value of the global variable part
'***************************************************
Sub PickPart(part)
    Peedy.Stop
    Peedy.Play "Alert"
    Select Case part
        Case 1
            Call top.Index.Part1
        Case 2
            Call top.Index.Part1_1
        Case 3
            Call top.Index.Part2
        Case 4
            Call top.Index.Part2_1
        Case 5
            Call top.Index.Part2_2
        Case 6
            Call top.Index.Part3
        Case 7
            Call top.Index.Part3_1
        Case 8
            Call top.Index.Part4
        Case 9
        Case 10
End Select
End Sub

'***********************************************************************
'Routines for Peedy.htm
'***********************************************************************
Sub CallPeedy()
    Peedy.Stop
    Peedy.Show
    Peedy.Speak "Here I am " & UsrName
    Peedy.Play "Blink"
End Sub

Sub Speak(text1)
    Dim checkText
    Peedy.Stop
    If text1 <> "" Then
        checkText = Sanitize(text1)
        If InStr(checkText, "bleep") Then
            Peedy.Play "Decline"
            Peedy.Speak "I'm sorry, I'd prefer not to say that."
        Else
            Peedy.Speak "You said: " & text1
        End If
    Else
        Peedy.Speak "If you enter some text in the box, I'll repeat what you said."
    End If
End Sub

Sub TellJoke()
    Dim jokeNum

    Randomize
    jokeNum = Int((7 * Rnd) + 1)
    Peedy.Stop

    Select Case jokeNum
        Case 1
            Peedy.Speak "Why couldn't the parrot talk to the English professor?"
            Peedy.Play "Think"
            Peedy.Play "Alert"
Peedy.Speak "Because he only spoke pigeon English."

Case 2
Peedy.Speak "What do you call a minor bird accident?"
Peedy.Play "Think"
Peedy.Play "Alert"
Peedy.Speak "A feather bender."

Case 3
Peedy.Speak "What do you call a formal dance for ducks?"
Peedy.Play "Think"
Peedy.Play "Alert"
Peedy.Speak "A fowl ball."

Case 4
Peedy.Speak "What's the difference between a miser and his pet canary?"
Peedy.Play "Think"
Peedy.Play "Alert"
Peedy.Speak "One's a little cheap, but the other's a little cheaper."

Case 5
Peedy.Speak "What did the parrot say when she fell in love with the frog?"
Peedy.Play "Think"
Peedy.Play "Alert"
Peedy.Speak "Polly wants a croaker!"

Case 6
Peedy.Speak "What would you name a parrot made out of plastic?"
Peedy.Play "Think"
Peedy.Play "Alert"
Peedy.Speak "Polly-Esther!"

Case 7
Peedy.Speak "Why does a chicken coop have only 2 doors?"
Peedy.Play "Think"
Peedy.Play "Alert"
Peedy.Speak "Because if it had 4, it would be a sedan!"
End Select
Peedy.Play "Pleased"
Peedy.Play "Idle1_1"
End Sub

'******************************************
'Routines for AboutPeedy.htm
'******************************************
Sub AboutPeedy()
    Dim display
    display = window.open("AboutPeedy.htm", "Index")
End Sub

Sub AboutSite()
    Peedy.Stop
    Peedy.Show
    Peedy.MoveTo window.screenLeft-300, window.screenTop+50
    Peedy.Play "Explain"
    Peedy.Speak "This site uses frames."
    Peedy.GestureAt window.screenLeft+100, window.screenTop+100
    Peedy.Speak "The frame to your right displays my home page. You can recognize it by it's yellow background."
    Peedy.Play "Blink"
    Peedy.Speak "As you can see, my home page has a text input box and several buttons that you can use to control and communicate with me."
    Peedy.GestureAt window.screenLeft, window.screenTop+200
    Peedy.Speak "The About Peedy button,"
    Peedy.MoveTo window.screenLeft+50, window.screenTop+200
    Peedy.GestureAt window.screenLeft, window.screenTop+200
    Peedy.Speak "loads the About Peedy page that you're currently viewing over here in the main display frame."
    Peedy.Play "Blink"
    Peedy.GestureAt window.screenLeft-300, window.screenTop+200
    Peedy.Speak "The About Peedy page describes how to control and communicate with me. Click a 'Narrate' button and I'll explain what to do."
    Peedy.Play "Idle1_2"
End Sub

Sub AboutTalk()
Peedy.Stop
Peedy.Show
Peedy.Play "Explain"
Peedy.Speak "There are two ways to talk to me."
Peedy.Play "Blink"
Peedy.Speak "Most of the time, if you simply click on me using your Left mouse button a dialog box will appear. Simply enter your question into the box and click OK or press the 'enter' key."
Peedy.GestureAt window.screenLeft+50, window.screenTop+20
Peedy.Speak "You can also talk to me by entering text into the box on my home page and clicking the 'Ask' button."
Peedy.GestureAt window.screenLeft-300, window.screenTop+400
Peedy.Speak "You'll find some example questions under the Talk to Peedy section of the About Peedy page."
Peedy.Play "Blink"
Peedy.Speak "Go ahead and ask me a question, or click on the 'Tell a joke' button and I'll tell you a joke."
Peedy.Play "Idle1_2"
End Sub

Sub AboutMove()
Peedy.Stop
Peedy.Show
Peedy.MoveTo window.screenLeft-400, window.screenTop+200
Peedy.Play "Explain"
Peedy.Speak "Sometimes, I may get in your way."
Peedy.Play "Blink"
Peedy.Speak "When I'm in the way, simply Left-click on me and drag me somewhere else on the screen."
Peedy.Play "RestPose"
Peedy.Speak "I really don't mind being moved. I'm here to assist you, not to be a nuisance."
Peedy.Play "Blink"
Peedy.Speak "Go ahead and move me now, just to see how I react."
End Sub

Sub AboutHide()
Peedy.Stop
Peedy.Show
Peedy.Play "Explain"
Peedy.Speak "Sometimes, you'll just want me to go away and leave you alone."
Peedy.Play "Blink"
Peedy.Speak "Don't worry, you won't hurt my feelings. I'm here to help you, not to annoy you!"
Peedy.Play "RestPose"
Peedy.Speak "When you want me to leave, just Right-click on me and select 'Hide' from the menu."
Peedy.MoveTo window.screenLeft-300, window.screenTop+100
Peedy.GestureAt window.screenLeft+50, window.screenTop+100
Peedy.Speak "If you want me to come back, just click on the 'Call Peedy' button on my home page and I'll reappear."
Peedy.Play "Blink"
Peedy.Speak "Go ahead and hide me now, then click on the 'Call Peedy' button to make me reappear."
End Sub

Sub AboutSpeak()
Peedy.Stop
Peedy.Show
Peedy.Play "Explain"
Peedy.Speak "I like to learn new words."
Peedy.GestureAt window.screenLeft+50, window.screenTop+20
Peedy.Speak "If you'd like to teach me something new to say, just enter the words in the box on my home page and click the 'Speak' button."
Peedy.Play "Blink"
Peedy.Speak "Go ahead and try it now."
End Sub

Sub AboutRepeat()
Peedy.Stop
Peedy.Show
Peedy.Play "Explain"
Peedy.Speak "Sometimes you may want me to repeat something that I said."
Peedy.Play "Blink"
Peedy.Speak "Just Right-click on me and select 'Repeat last statement' from the menu. Go ahead and try it now."
End Sub

Sub AboutStop()
Peedy.Stop
Peedy.Show
Peedy.Play "Explain"
Peedy.Speak "Sometimes you'll want to make me stop whatever it is I'm doing."
Peedy.Play "Blink"
Peedy.Speak "To make me stop, just Right-click on me and select 'Stop' from the menu."
Peedy.Play "RestPose"
Peedy.Speak "To practice, I'll start reading a magazine. Once I've started, go ahead and stop me."
Peedy.Play "Reading"
stopDemo = 1
End Sub

Sub AboutOptions()
    Peedy.Stop
    Peedy.Show
    Peedy.Play "Explain"
    Peedy.Speak "My Advanced Character Options menu provides a few more controls."
    Peedy.MoveTo window.screenLeft-400, window.screenTop+200
    Peedy.Speak "To access this menu, simply Right-click on me and select 'Advanced Character Options'. Go ahead and do this now."
    Peedy.Play "Think"
    Peedy.Speak "Probably the most valuable option here is the 'Speaking speed' control."
    Peedy.MoveTo window.screenLeft+50, window.screenTop+200
    Peedy.Speak "You'll see this under the 'Output' tab. To adjust the speed of my speech, simply drag the grey arrow in the appropriate direction. Click OK when you're done."
    Peedy.Play "RestPose"
    Peedy.Speak "The other controls here are pretty self-explanatory. If you can't hear me speak or don't have a text-to-speech engine installed, many of these controls won't be relevant to you."
End Sub
AVStutorial.vbs

' AVStutorial.vbs
' Created: 3-27-03
' Last updated: 4-30-03

'************************
'Global variables
'************************
Dim Peedy
Dim winFeat, part, tutorWin

Set Peedy = top.Peedy.Agent.Characters("Peedy")
Peedy.Get "animation", "Congratulate", False
winFeat =
"top=10,left=0,height=500,width=800,status=yes,toolbar=yes,
menubar=yes,location=yes,resizable=yes,scrollbars=yes"

Sub Window_OnLoad
   On Error Resume Next
   top.window.moveTo 0,0
   top.window.resizeTo screen.availWidth,
screen.availHeight
   Peedy.Commands.Add "Replay", "Replay latest segment"
   Peedy.Commands.Add "ExitTut", "Exit tutorial"
   top.Peedy.part = 0
   top.Peedy.drag = 0
   top.Peedy.click = 2
   Peedy.Stop
   Peedy.Show
   Peedy.Play "Wave"
   Peedy.Speak "Welcome to the Wolf keyword searching
tutorial."
   Peedy.Play "Explain"
   Peedy.Speak "This tutorial will introduce you to the
keyword searching features available in the Wolf online
catalog."
   Peedy.GestureAt window.screenLeft+70,
   window.screenTop+150
   Peedy.Speak "To get started, click on the 'Begin'
button under Part 1."
   Peedy.Play "Blink"
   Peedy.Play "Idle1_1"
   top.Peedy.part = 1
End Sub
Sub Window_OnUnLoad
    On Error Resume Next
    Peedy.Stop
    Peedy.Commands.Remove "Replay"
    Peedy.Commands.Remove "ExitTut"
    top.Peedy.drag = 1
    top.Peedy.click = 1
    top.Peedy.part = 0
    top.window.moveTo 20,20
    top.window.resizeTo screen.availWidth-100,
    screen.availHeight-30
End Sub

Sub InitCommands
    Set tutorWin =
    window.open("http://library.wou.edu/search/X", "Tutorial",
    winFeat)
    Peedy.Stop
    Peedy.Show
End Sub

'*******************************
'Phrase searching tutorial
'*******************************

Sub Part1()
    InitCommands
    On Error Resume Next
    Peedy.MoveTo window.screenLeft+600,
    window.screenTop+30
    Peedy.Play "Explain"
    Peedy.Speak "This is the Wolf keyword search page."
    Peedy.Play "Blink"
    Peedy.Speak "If you simply enter multiple words, Wolf will
    search for them exactly as you entered them. This is called \emp\'Phrase searching.\""
    Peedy.Play "Blink"
    Peedy.GestureAt window.screenLeft+300,
    window.screenTop+150
    Peedy.Speak "To see how this works, enter the phrase 'human nature' into the search box and click the 'Search' button."
    Peedy.Play "Idle1_2"
    Peedy.Speak "When you're ready to continue with the tutorial, just click on me."
    Peedy.Play "Blink"
    Peedy.Play "Idle1_1"
    top.Peedy.part = 2
End Sub

Sub Part1_1()
On Error Resume Next
    Peedy.Stop
    Peedy.Show
    Peedy.MoveTo window.screenLeft+150, window.screenTop+100
    Peedy.GestureAt window.screenLeft+75, window.screenTop+30
    Peedy.Speak "Wolf will tell you how many records contain the phrase you entered."
    Peedy.GestureAt window.screenLeft+200, window.screenTop+200
    Peedy.Speak "And give you a list of titles representing the search results."
    Peedy.Play "Blink"
    Peedy.Speak "To view a detailed record for a specific title, simply click on the title."
    Peedy.GestureAt window.screenLeft+200, window.screenTop+200
    Peedy.Speak "Go ahead and click on one of the titles now."
    Peedy.Play "Idle1_1"
    Peedy.MoveTo window.screenLeft+150, window.screenTop+20
    Peedy.GestureAt window.screenLeft+200, window.screenTop+200
    Peedy.Speak "When you view the detailed record, instances of the phrase you searched appear in red."
    Peedy.MoveTo window.screenLeft+600, window.screenTop+100
    Peedy.Play "Congratulate"
    Peedy.Speak "Congratulations! You've now completed the Phrase searching tutorial."
    Peedy.Play "Blink"
    Peedy.Speak "Try a couple of searches on your own for practice. One of my favorites is 'artificial intelligence'."
    Peedy.Play "Pleased"
    Peedy.GestureAt window.screenLeft+300, window.screenTop+150
    Peedy.Speak "To do another search, simply type another phrase into the search box on this page."
    Peedy.Play "Blink"
Peedy.Speak "Click on me when you're ready to continue with the next part of the keyword searching tutorial. Or, close the Wolf window and return to the tutorial page."
Peedy.Play "Idle2_1"
  top.Peedy.part = 3
End Sub

'************************************
'Boolean operator searching tutorial
'************************************

Sub Part2()
InitCommands
On Error Resume Next
  Peedy.Play "Wave"
  Peedy.Speak "Welcome back."
  Peedy.MoveTo window.screenLeft+600, window.screenTop+30
  Peedy.Play "Explain"
  Peedy.Speak "In this tutorial, I'll teach you how to use Boolean operators in Wolf keyword searches."
  Peedy.Play "Blink"
  Peedy.Play "RestPose"
  Peedy.Speak "Often, you will want to search for records that contain more than one term, but not necessarily as an exact phrase."
  Peedy.Play "Blink"
  Peedy.Speak "To do this type of searching, Wolf requires that you use the Boolean operator 'and' to combine your search terms"
  Peedy.Play "RestPose"
  Peedy.Speak "For example, let's search for records that contain the term 'dog' and the term 'cat'."
  Peedy.GestureAt window.screenLeft+300, window.screenTop+150
  Peedy.Speak "Enter 'dog\emp\and cat' into the keyword search box. Then, click the Search button."
  Peedy.Play "Idle1_2"
  Peedy.Play "Think"
  Peedy.MoveTo window.screenLeft+150, window.screenTop+100
  Peedy.GestureAt window.screenLeft+75, window.screenTop+30
  Peedy.Speak "Notice how many records this search found."
  Peedy.GestureAt window.screenLeft+300, window.screenTop+150
Peedy.Speak "To compare these results with the results of a Phrase search, simply delete the "and" operator from the search box and click the 'Search' button."
Peedy.Play "Idle1_1"
Peedy.Play "Think"
Peedy.GestureAt window.screenLeft+75, window.screenTop+30
Peedy.Speak "You should notice that you retrieved a lot fewer records when you omitted the "and" from your search. That's what usually happens."
Peedy.Play "Explain"
Peedy.Speak "When the 'and' operator is omitted, Wolf only retrieves records that contain the exact phrase 'dog cat', which doesn't occur very often."
Peedy.Play "RestPose"
Peedy.Speak "When you include the "and" operator, Wolf will retrieve records that contain both terms, no matter how they appear in relation to one another."
Peedy.GestureAt window.screenLeft+300, window.screenTop+150
Peedy.Speak "Go ahead and try using the "or" operator in a couple more searches on your own. Click on me when you're ready to continue with the tutorial."
Peedy.Play "Idle2_1"
top.Peedy.part = 4
End Sub

Sub Part2_1()
On Error Resume Next
Peedy.Stop
Peedy.Show
Peedy.MoveTo window.screenLeft+150, window.screenTop+100
Peedy.Speak "Let's try another search, this time using the Boolean operator "or" to combine our search terms."
Peedy.Play "Explain"
Peedy.Speak "'Or' is especially useful when you need to search terms that have one or more commonly used synonyms."
Peedy.Play "Blink"
Peedy.Speak "For example, say you wanted to search for books about movies."
Peedy.GestureAt window.screenLeft+300, window.screenTop+150
Peedy.Speak "Go ahead and enter the term 'movies' in the search box and click the 'Search' button."
Peedy.Play "Idle1_3"
Peedy.Play "Think"
Peedy.GestureAt window.screenLeft+75,
window.screenTop+30
Peedy.Speak "Notice how many records were found."
Peedy.Play "Blink"
Peedy.GestureAt window.screenLeft+300,
window.screenTop+150
Peedy.Speak "Now, do a search for the term 'films'."
Peedy.Play "Idle1_1"
Peedy.Play "Think"
Peedy.GestureAt window.screenLeft+75,
window.screenTop+30
Peedy.Speak "Again, notice how many records were
found."
Peedy.Play "Blink"
Peedy.GestureAt window.screenLeft+300,
window.screenTop+150
Peedy.Speak "Now, add \emp\'or movies\' to your search
and click the 'Search' button."
Peedy.Play "Idle1_3"
Peedy.Play "Think"
Peedy.GestureAt window.screenLeft+75,
window.screenTop+30
Peedy.Speak "Notice how many records were found this
time."
Peedy.Play "Blink"
Peedy.Speak "As you can see, if you search either of
these terms alone, you might miss many relevant books."
Peedy.Play "Explain"
Peedy.Speak "By combining terms together using
\emp\'or\', you found records that contain either relevant
term in a single search."
Peedy.MoveTo window.screenLeft+600,
window.screenTop+100
Peedy.GestureAt window.screenLeft+300,
window.screenTop+150
Peedy.Speak "Try a couple of searches on your own for
practice. Try using \emp\'and\' and \emp\'or\' to combine the
same terms and compare the results."
Peedy.Play "Blink"
Peedy.Speak "Click on me when you're ready to continue
with the tutorial."
Peedy.Play "Idle2_1"
top.Peedy.part = 5
End Sub
Sub Part2_2()
On Error Resume Next
  Peedy.Stop
  Peedy.Show
  Peedy.MoveTo window.screenLeft+150, screenTop+100
  Peedy.Speak "Wolf supports one more Boolean operator called "And Not""
  Peedy.Play "Explain"
  Peedy.Speak "And Not' is most useful when you want to search for a term that has different meanings when it's used in different contexts."
  Peedy.Play "Blink"
  Peedy.Speak "Say, for example, you wanted to find books about the language Latin."
  Peedy.GestureAt window.screenLeft+300, window.screenTop+150
  Peedy.Speak "Enter the term 'Latin' in the search box and click the 'Search' button."
  Peedy.Play "Idle1_3"
  Peedy.Play "Think"
  Peedy.GestureAt window.screenLeft+75, window.screenTop+30
  Peedy.Speak "Notice that Wolf found a large number of books,"
  Peedy.GestureAt window.screenLeft+200, window.screenTop+200
  Peedy.Speak "But many of them are about Latin America, rather than the language Latin."
  Peedy.Play "Think"
  Peedy.GestureAt window.screenLeft+300, window.screenTop+150
  Peedy.Speak "Now, add 'and not America' to your search and click the search button."
  Peedy.Play "Idle1_4"
  Peedy.Play "Think"
  Peedy.GestureAt window.screenLeft+75, window.screenTop+30
  Peedy.Speak "Notice that Wolf found many fewer records this time,"
  Peedy.GestureAt window.screenLeft+200, window.screenTop+200
  Peedy.Speak "And, fewer of them are about Latin America."
  Peedy.Play "Blink"
  Peedy.MoveTo window.screenLeft+600, window.screenTop+100
  Peedy.Play "Congratulate"
Congratulations! You've now completed the Boolean searching tutorial.

Peedy.GestureAt window.screenLeft+300, window.screenTop+150
Peedy.Speak "If you'd like, browse your search results and try a couple more searches on your own for practice."
Peedy.Play "Blink"
Peedy.Speak "When you're ready to continue with the next part of the keyword searching tutorial, click on me, or you can close the Wolf window and return to the tutorial page."
Peedy.Play "Idle2_1"
top.Peezy.part = 6

Truncation and wildcard searching tutorial

Sub Part3()
InitCommands
On Error Resume Next
Peedy.Play "Wave"
Peedy.Speak "Welcome back."
Peedy.MoveTo window.screenLeft+600, window.screenTop+30
Peedy.Play "Explain"
Peedy.Speak "In this tutorial, I'll teach you how to use truncation and wildcard symbols in Wolf keyword searches."
Peedy.Play "Blink"
Peedy.Play "RestPose"
Peedy.Speak "When you do a keyword search, Wolf searches only for the exact term you enter."
Peedy.Play "Blink"
Peedy.Play "RestPose"
Peedy.Speak "But in English, many words will take slightly different forms depending upon how they are used."
Peedy.Play "RestPose"
Peedy.Speak "Truncation and wildcard symbols allow you to tell Wolf to include minor variations of a term in your search."
Peedy.Play "Think"
Peedy.Speak "For example, say you wanted to search for books about children."
Peedy.Play "Blink"
Peedy.Speak "If you enter just the term 'children', Wolf will only retrieve records that contain that exact term."
Peedy.Play "Blink"
Peedy.Speak "Wolf would not retrieve records that only contain the term 'child', even though many of these books would probably be relevant to your needs."
Peedy.Play "RestPose"
Peedy.Speak "You can use an asterisk to expand your search to include both the term 'child' and the term 'children'."
Peedy.GestureAt window.screenLeft+300, window.screenTop+150
Peedy.Speak "Enter 'child*' in the search box. Click the 'Search' button to submit your search."
Peedy.Play "Idle1_2"
Peedy.Play "Think"
Peedy.GestureAt window.screenLeft+200, window.screenTop+400
Peedy.Speak "Take a minute to examine some of the records retrieved in this search. Click on me when you're ready to continue."
Peedy.Play "Idle2_1"
top.Peedy.part = 7
End Sub

Sub Part3_1()
On Error Resume Next
Peedy.Stop
Peedy.Show
Peedy.Speak "A single asterisk expands your term up to five characters beyond the base term."
Peedy.Play "Blink"
Peedy.Speak "Appending two asterisks allows for unlimited expansion."
Peedy.Play "Blink"
Peedy.Play "RestPose"
Peedy.Speak "If you need to expand based on a character within a word, rather than at the end of a word, use a question mark '?' in place of the character that varies."
Peedy.Play "Think"
Peedy.Speak "For example, you can search for both 'woman' and 'women' in a single search."
Peedy.GestureAt window.screenLeft+300, window.screenTop+150
Peedy.Speak "Enter 'wom?n' in the search box and click the 'Search' button."
Peedy.Play "Idle1 2"
Peedy.Play "Think"
Peedy.Play "Congratulate"
Peedy.Speak "Congratulations! You have now completed the Wolf Keyword Searching Tutorial."
Peedy.Play "Blink"
Peedy.GestureAt window.screenLeft+200, window.screenTop+400
Peedy.Speak "Take a minute to examine some of the records retrieved in this search, or continue practicing your new keyword searching skills."
Peedy.Play "Explain"
Peedy.Speak "If you have questions about searching Wolf, please ask at the Information desk or call (503) 838-8418."
Peedy.Play "Blink"
Peedy.Speak "When you're done, simply close the Wolf window and return to the tutorial page."
Peedy.Play "Wave"
Peedy.Speak "See you later!"
Peedy.Hide
'
Peedy.Speak "When you're ready to start the next part of the keyword searching tutorial, click on me, \Pau=20\or you can close the Wolf window and return to the tutorial page."
'  top.Peedy.part = 8
End Sub

'Sub Part4()
'InitCommands
'  On Error Resume Next
'  Peedy.Speak "Welcome back."

'End Sub

'Sub Part5()
'InitCommands
'  On Error Resume Next
'  Peedy.Speak "Welcome back."
'End Sub