The EUSES Web Macro Scenario Corpus, Version 1.0

Christopher Scaffidi[1], Allen Cypher[2], Sebastian Elbaum[3],
Andhy Koesnandar[3], Brad Myers[4]

November 2006
CMU-HCII-06-105

Human-Computer Interaction Institute
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA 15213

[1]Institute for Software Research, Carnegie Mellon Univ., Pittsburgh, PA 15213
[2]Almaden Research Center, IBM, San Jose, CA, 95120
[3]Computer Science and Engineering Department, Univ. of Nebraska, Lincoln, NE 68588
[4]Human-Computer Interaction Institute, Carnegie Mellon Univ., Pittsburgh, PA 15213

Abstract

Web macros use the programming-by-example concept to automate user actions within a web browser. Although web macro recorders and players have grown in sophistication over the past decade, we believe that these tools cannot yet meet the needs of real users. Based on observations of browser users, we have compiled various scenarios describing tasks that end users would benefit from automating using web macros. Our analysis of these scenarios yields specific requirements that web macro tools must support if those tools are to be applicable to real-life situations. For example, these opportunities for improvement include better support for triggering macros on events, authenticating to sites, transporting data to/from spreadsheets, taking advantage of data's semantics, and recovering from errors. Our collection of requirements constitutes a benchmark for evaluating new and improved web macro tools. We developed this corpus as a collaboration within the EUSES Consortium, whose aim is to help End Users Shape Effective Software.

This work was funded in part by the National Science Foundation under ITR grant CCF-0325273 (via the EUSES Consortium) and by the National Science Foundation under ITR grants CCF-0438929 and CCF-0324861.

Keywords: web, macros, browser, end user programming, end user software engineering, programming by example, programming by demonstration

1. Background and Motivation

Performing repetitive actions in a browser is tedious. Therefore, since 1997 researchers have provided tools enabling end users to automate web browser actions [14][23]. For nearly a decade, successive tools have appeared, most of which have followed the programming-by-example (PBE) paradigm that has been previously applied in a variety of environments ranging from spreadsheets to HyperCard [2][10]. In this paradigm, a macro recorder watches the user perform the operations and tries to determine the user's intent; the recorder may ask for additional examples or information in order to significantly minimize or eliminate inference while determining intent. It then generates a macro as a representation of the user's intent (generally as a sequence of steps). Later, a macro player executes the macro by processing the macro and acting on new data in a way consistent with the user's intent.

Web macros can be helpful in other ways, in addition to automating daily tasks. For example, when performing a procedure in a web application is complex and hard-to-learn, users who have mastered that difficult procedure could create a macro as a teaching tool for other users, in order to encapsulate and communicate the steps required to perform the procedure. In addition, web application developers can use macros to create automated test suites. Finally, macros provide a platform for creating new applications through screen scraping / mashups, and for creating assistive technologies.

Given these benefits, it might seem as though these tools should be in widespread use by now, particularly among information workers. After all, web browsers are part of the fabric of life, as over 50% of Americans use the Internet [15], and over 50% of American workers use computers at work [21]. Moreover, office work involving data manipulation, particularly within browsers, is highly repetitive and seems perfect for automation [22]. In addition, over 40% of respondents to a recent survey of information workers reported that they or their subordinates recorded *spreadsheet* macros in the past 3 months [19], so users seem to be generally capable of understanding the process of recording and replaying macros.

Yet despite these factors, web macros do not seem to be in widespread use. One hypothesis for this is that current web macro tools are not capable of automating the tasks that end users perform with their browsers.

To delineate the requirements that macro tools must satisfy in order to be more useful to end users, we analyze various real-world tasks that should ideally be automatable with web macros. We have selected these tasks because automating them would offer clear benefits to end users. For example, automating one time-consuming task was so desirable to one end user that he paid a professional PHP/Perl programmer to automate the task for him; two other tasks were automated by open source programmers in order to benefit other people. For certain tasks, we have observed our co-workers manually performing actions over and over; automating these tasks would offer significant time benefits. Finally, some tasks have been performed repetitively by us, and we would very much like to automate these tasks to save ourselves time, but we have no suitable PBE macro tool to use.

We describe these tasks in a scenario format, analyze the scenarios, and identify a list of requirements that can serve as a benchmark for measuring improvement in the real-world usefulness of macro tools. The requirements that we have identified fall into the following broad categories:

Triggering macros: Most macro tools allow users to execute macros on demand through menu items or buttons, and some macro players can execute macros on a schedule. However, most PBE macro recorders lack support for more sophisticated script triggers, such as automatically executing macros any time that a user visits a page. If users want to use these triggers, they must resort to non-PBE tools [13], which require the user to learn a full-fledged programming language.

Using objects on web pages: Before the macro player can read data from a widget, paragraph, or other element on the web page, the player must first find the object. (The same is true for filling in forms and performing other manipulations within the web page.) This challenge is particular to web macro tools because unlike other applications that have been automated with PBE, web applications can return different HTML at different times, even for the same http request, and page layout can change radically as

webmasters modify functionality. According to the Internet Archive's Wayback Machine,[1] such evolution has happened with many sites involved in our scenarios. Consequently, when macro players apply the user's intent in the runtime context, they must do far more than just blindly plod through a set of operations (as with traditional programs). Instead, players typically use heuristics to adapt to changing content [14][18][23]. However, these tools are unable to adapt when changes fall outside of existing heuristics. In addition, tools cannot yet adapt to wholesale URL changes, whereas humans can cope quite well with messages like "This page has moved to http://foo.com/bar.jsp -- Please update your links."

Reading/writing data outside pages: The earliest macro tools supported exporting portions of web pages to files [23], but that basic functionality did not enable users to read data from files then send it to servers (through form fields). Later tools now support reading data from one web site and sending it to another site, but tools should also support reading and writing data from a variety of file formats and locations, including XML files and spreadsheets, as this is a requirement for tasks that system administrators and other users commonly need to perform [6]. Finally, macros should be able to read data from user-defined parameters using values entered prior to runtime (and reused over several executions), or parameters manually entered by the user at runtime.

Transforming data: Perhaps the greatest strength of macro recorders is their ability to scrape content from the web. However, data items often require reformatting or transformation before they can be reused in another context. Some macro tools allow users to create simple rules that transform the data, typically through a spreadsheet-like formula or a programming language [4][6]. However, users still need better support to "build formulas for complex structure analysis, and … identify elements within structured data without having to write code" [4].

Executing control structures: Although support for conditionals and loops was lacking in the first PBE tools [14][23], it has been added to most later tools. Yet in the web macro domain, at present, web macro tools have limited support for conditionals; one exception is Robofox, though users must still manually insert conditionals after recording macros [8]. Other tools such as Creo support loops that operate "foreach" item in a list on a web page [3]; more sophisticated types of loops such as "while(condition)" are generally not supported in current web macro tools.

Recovering from failure: Perhaps the thorniest category of requirements relates to detecting failure and recovering appropriately. At least one tool now supports assertion checking, which helps to detect failure [8]. However, recovering after failure is complicated if the failed macro modified some data on the web prior to failure. Unlike databases, web macro tools and web servers currently do not support transactions, so rolling back work is not possible.

Supporting macro maintenance: The macro tool should represent macros in a form that enables users to evaluate, adapt, and debug macros. Some macro tools, such as [8], represent macros' internal structure in a way that enables the user to understand what steps each macro contains and to delete steps if desired (which presumably will also facilitate sharing of macros). However, many other tools lack such features. In addition, maintainability is impeded by a lack of basic debugging services. For example, trace and breakpoint features might be useful when trying to understand why a script has stopped functioning correctly after a change in a web site's structure.

The requirements we have identified comprise a benchmark to measure improvements in macro tools. This benchmark's benefits are similar to those of the Test Suite for Programming by Demonstration [17]. First, our benchmark illustrates the wide range of potential applications for PBE web macros. Second, it enables researchers to test their tools with tasks from the real world.

As was the case with the Test Suite for Programming by Demonstration, we do not claim that our list is complete in the sense that satisfying these requirements will make macro tools absolutely perfect for all imaginable tasks. Instead, the benchmark constitutes a seed that can grow into a larger corpus as other

[1] Internet Archive, `http://www.archive.org/`

researchers contribute additional scenarios describing real-world situations where PBE web macros would be beneficial.

We have chosen to include the word "EUSES" in our corpus title for two reasons. First, we are members of the EUSES Research Consortium[2], and within our community, this corpus represents a shared vision of what web macro tools should be capable of achieving in the future. Second, the word "EUSES" is an acronym for "End Users Shaping Effective Software," which is consistent with the ultimate goal of our scenario corpus: to foster the development of tools that will help users create more effective and useful web macro software.

In this report, we develop our corpus in two stages. First, in Sections 2 and 3, we describe each situation as a scenario involving repetitive actions that users would benefit from automating with a macro. Second, in Section 4, we analyze these scenarios to identify specific requirements that web tools must satisfy in order to be useful for real-world tasks.

2. Overview of Scenarios

Each scenario represents a task that end users would benefit from automating. In most cases, we have actually observed users performing these tasks. In a few other cases, the task has already been automated with a script, and we have reverse-engineered the script to describe the task that the user wanted to perform. In all cases, automating the task would yield significant time savings to the user.

Each task involves a pre-condition and a post-condition: the scenario goal is to "get from here to there." For most scenarios, users currently must achieve this manually using the actions described in each scenario. We hope to foster innovative development of tools that will someday be able to achieve the same effects automatically.

In pursuit of this goal, different macro tools may take different implementation approaches. For example, some existing macro players are agents that emulate a browser, while others are toolbars that manipulate the browser like a puppet.

Therefore, while our scenario descriptions highlight *what* requirements must be satisfied by tools, we do not specify *how* these requirements must be satisfied. We believe that specifying implementation details would be fruitless (although in Section 4, we do offer some ideas for implementations).

Indeed, we do not even stipulate the specific examples that macro recorders may request from users: if a macro recorder can do better by requiring more information, then that innovation represents a valid tradeoff worth considering. For example, some macro recorders use a pure PBE approach, while others allow users to augment the macro with procedural code [4].

2.1 Selection Criteria for Scenarios

End users perform a plethora of repetitive tasks, so selecting just a few for detailed analysis requires applying a few judicious criteria, as follows.

First, our ultimate goal is to enable people to use web macros for real-life tasks. Consequently, we have shied away from presenting hypothetical scenarios and, as much as possible, focused on real situations encountered by users in actual practice. These are not abstract situations, but rather "instantiated" tasks grounded in concrete user experiences.

Second, we want to provide a benchmark to measure the improvement of macro tools. Therefore, we have chosen scenarios that highlight the ground that researchers have yet to cover, rather than scenarios that users could easily automate using existing PBE web macros. Most scenarios describe repetitive actions that

[2] http://eusesconsortium.org/

users still must manually perform, though a few describe repetitive actions that have been automated with scripts.

Last, in order to illustrate the breadth of macros' applicability, we have selected scenarios involving several types of users. These include office workers, online shoppers, financial analysts, IT staff, and others. Moreover, these scenarios come from a variety of sources:

- *Our own experience:* Two scenarios were performed by us in our role as end users. Like all researchers, we lead double lives. On one hand, we are experienced computer users who can program in a variety of languages when necessary. On the other hand, we are also end users: We are constrained by time and interest, so we must often live within the confines of existing applications rather than writing our own. (Moreover, because we typically lack access to existing applications' source code, we cannot easily change them.)
- *Contextual inquiry:* Three scenarios were uncovered by our recent contextual inquiry of information workers [22]. We observed the work of 3 administrative assistants, 4 office managers, and 3 webmasters/graphic designers at Carnegie Mellon University. We watched each for one to three hours, in some cases spread over two days, and used a tape recorder and notebook to record information.
- *Co-workers:* Two scenarios were performed by our co-workers. We observed these repetitive actions during the course of work and only later realized that they were suitable for web macro automation.
- *Online sources:* Three scenarios involving screen-scraping were automated by people using scripting languages. In two cases, the programmer publicly posted the script (one PHP and one JavaScript); we have reverse-engineered these scripts into macro scenarios. In the third case, an end user publicly posted a specification for the scenario (which probably was implemented in Python, PHP or Perl); we have converted this specification into a macro scenario.

2.2 Structure of Scenarios

In Section 3, we use a specific pattern to describe each scenario. We designed this pattern with minimalism in mind so other researchers could easily follow our pattern and contribute new scenarios.

Each scenario consists of several sections. If other researchers follow this section structure when augmenting our corpus with new scenarios, then readers may be able to easily locate information. Extra sections might be added for specific scenarios to contain information that does not readily fit into any standard section.

The sections are as follow, with required sections flagged by asterisks:

*	*Title:*	This makes it easy for researchers to refer to scenarios.
*	*Typical User:*	This makes it easy for readers to find scenarios that might be performed by specific populations.
*	*Overview:*	This explains the context and forces that would prompt a *Typical User* to perform the scenario.
*	*Starting Conditions:*	This identifies pre-conditions that hold true before the scenario.
*	*Result:*	This identifies post-conditions that should hold true after the scenario. This specifies the goal that a user or macro must achieve in order to be judged capable of performing this scenario.
*	*Actions:*	This describes the algorithm, process, or steps that the user performs (or that the user performs with the help of a tool). This is *not* to suggest that macros must perform the same algorithm. A macro need only achieve the *Result* in order to be considered successful.
	Action Details:	This could include screenshots, snippets of HTML, video, or other pieces of information that make the A*ctions* clearer.
	Variations:	This discusses ways in which users might tweak the *Actions, Starting Conditions,* or *Results* into a slightly different but similar scenario.
	Macro Maintenance:	This examines how the scenario might evolve over time, prompting changes to macros that attempt to automate the scenario.
*	*Scenario Source:*	This summarizes the observations or other empirical data that generated this scenario. This information helps communicate the extent to which this scenario represents the real world rather than a hypothetical situation.

3. Catalog of Scenarios

To summarize the linkage between the scenarios discussed in this section and the requirements discussed in the next section, we have provided the table below. This table indicates the scenarios that led us to each requirement. For detailed information on requirements, refer to corresponding subsections of Section 4.

An additional benefit of this table is that it highlights requirements that are required to support many scenarios. For example, Reading/Writing Data in the form of Text snippets (4.3.1) is part of every scenario and is therefore an essential feature of any macro tool. Because there are so many ways in which each scenario can break and require maintenance, we have marked each scenario's box in this table for the rows that deal with exception handling and maintenance. For details, refer to those subsections of Section 4.

Category - Requirements	**Scenario Title** **Typical user**	**Currency Converter** **Office worker**	**Package Tracker** **Online shopper**	**Path to Procurement** **Office worker**	**Peoplesoft Scraper** **IT staff**	**Per Diem Lookup** **Office worker**	**Person Locator Scraper** **Volunteer developer**	**Scraper for CMS** **Webmaster**	**Staff Lookup** **Office worker**	**Stock Analysis** **Financial analyst**	**Watcher for eBay** **Online shopper**
4.1 Triggering macros											
4.1.1 On-demand execution		X	X	X	X	X	X	X	X	X	
4.1.2 Scheduled execution							X				
4.1.3 Event-based triggers		X									X
4.1.4 Subroutines		X					X	X			
4.2 Using objects on web pages											
4.2.1 Text snippets		X	X	X	X	X	X	X	X	X	X
4.2.2 Tabular information			X	X	X	X	X	X	X	X	
4.2.3 Other HTML structures											X
4.2.4 Adaptation to changing page layout		X	X			X		X		X	X
4.2.5 Web form widgets		X	X	X	X	X		X	X	X	X
4.2.6 Adaptation to changing form fields		X	X	X						X	X
4.2.7 Adaptation to changing URLs						X					
4.3 Reading/writing data outside pages											
4.3.1 Browser APIs		X				X					
4.3.2 Spreadsheets and other files		X	X	X	X		X		X	X	
4.3.3 Parameters containing user input		X		X	X				X		
4.4 Transforming data											
4.4.1 Reformat to equivalent value		X				X	X		X	X	
4.4.2 Extraction of values' parts						X				X	
4.4.3 Combination of values						X				X	X
4.5 Executing control structures											
4.5.1 Looping operations			X	X	X	X	X	X	X	X	X
4.5.2 Conditional operations			X	X		X	X	X	X	X	
4.6 Recovering from failure											
4.6.1 Partial restarts				X			X	X		X	
4.6.2 Exception handlers		X	X	X	X	X	X	X	X	X	X
4.7 Supporting macro maintenance											
4.7.1 User-understandable representation		X	X	X	X	X	X	X	X	X	X
4.7.2 Editable macros		X	X	X	X	X	X	X	X	X	X
4.7.3 Features for debugging		X	X	X	X	X	X	X	X	X	X
4.7.4 Maintenance at runtime		X	X	X	X	X	X	X	X	X	X

Currency Converter — Typical User: office worker

Overview
When submitting an expense report for foreign travel, employees must convert each expense into US dollars using the currency conversion rate that was applicable on the date of the expense. During our contextual inquiry, several workers used the Actions below to perform this conversion.

Starting Conditions
The user knows the foreign denomination for the expense and has a spreadsheet with one row for each expense containing:

- The expense date
- The expense amount, in the foreign denomination

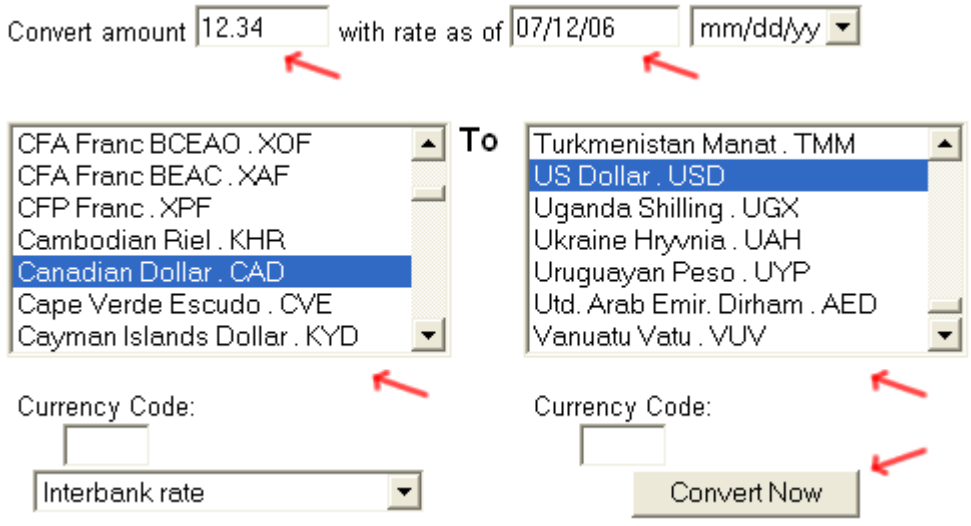
Result
Each expense row in the spreadsheet contains a third cell that contains the expense amount in US dollars.

Actions
1. For each row in the spreadsheet,
   1.1. Open the currency conversion web site.
   1.2. Submit the amount, date, foreign currency type, and target currency.
   1.3. Retrieve the result in US dollars.

Action Details

1.1 The site is publicly accessible at `http://www.oanda.com/converter/classic`

1.2 Submit the amount, date, foreign currency type, and target currency.



1.3 Retrieve the result in the target currency.

Variations

Suppose that a user is shopping at an online store that is physically located in a foreign country. In that case, the user might see prices in foreign currency and want to know what the corresponding prices are in American dollars. It would be convenient if the user could just highlight a price, right-click (to show a context menu), and select "Convert to US dollars." This could run a macro that looks at the site's URL, infers the correct currency for the URL's top-level domain, and then performs Actions like those above to convert the price to US dollars using today's conversion rate. The macro player might display the result in a popup window.

For a given expense report, the foreign currency is the same for each expense. However, the foreign currency is not recorded anywhere in the source spreadsheet—it is in the user's head. However, the foreign currency does vary from expense report to expense report. If the macro only needs to be usable for conversions from a single currency, then the source currency can be hard-coded in a macro; however, a variation that makes the macro reusable for multiple currencies would involve parameterizing the source currency in a way that allows the user to specify a source currency at runtime.

Unless the spreadsheet already contains the date in `MM/DD/YYYY` format, it is necessary to reformat the date to this format. Another variation is to reformat the date into one of the other formats supported by this web application (`DD/MM/YY` and `YY/MM/DD`), then set the rightmost dropdown accordingly.

Macro Maintenance

According to the Internet Archive's Wayback Machine, this currency converter form had exactly the same widgets in 1998 (when the page apparently appeared) as it does now. However, some of the options within the widgets have changed. For example, the internal code for the "Bulgiarian Lev" has changed from "BGL" to "BGN", and the "United Kingdom Pound" no longer appears at all. (Of course, some options have been added.) These changes could prompt maintenance activities to a macro automating this scenario.

The colors and fonts of text surrounding the form have changed several times in the past few years. Although the Wayback Machine does not contain any archived versions of the output page (Action 1.3), it seems likely that the style of this output page stayed in step with the style of the main web form. This evolution in style could force maintenance to a macro that relies on the color and font of the results in order to pick out currency amounts.

Scenario Source

Several administrative assistants performed this scenario during our contextual inquiry.

Package Tracker — Typical User: online shopper

Overview
Christmas shopping is moving online, creating a new concern for shoppers: making sure that gifts arrive before Christmas morning. Most online stores now email tracking numbers to customers as soon as the shipping company picks up the shipments. However, if the customers have ordered many shipments, perhaps from multiple sites, then checking the delivery dates for these shipments requires a great deal of repetitive effort, as described in the scenario below.

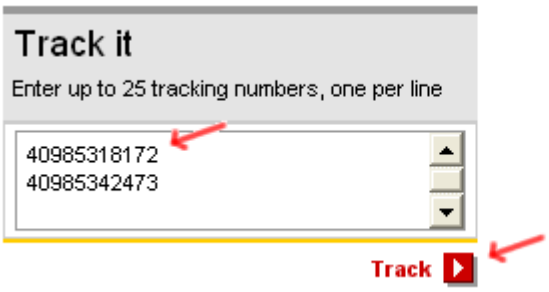
Starting Conditions
The user has a spreadsheet containing tracking numbers, one per line.

Result
For each shipment, the spreadsheet contains the corresponding shipment status alongside the tracking number.

Actions
1. While some shipments remain to be looked up,
   1.1. Open the DHL web site.
   1.2. Submit up to 25 tracking numbers.
   1.3. For each tracking number,
      1.3.1. Retrieve the corresponding shipment's status.

Action Details
1.1 The page is publicly accessible at `http://www.dhl-usa.com/home/home.asp`

1.2 Submit up to 25 tracking numbers.



1.3 For each tracking number, retrieve the corresponding shipment's status.

Tracking number	Status	Ship Date	Origin	Destination
40985318172	✓ Shipment delivered. Signed by LD FD -15 WAS 5/26/2006 11:14 am		LAKE MARY, FL	EAST WINDSOR, NJ
40985342473	✓ Shipment delivered. Signed by LD LOF -5510 FORBES AVE 6/3/2006 10:26 am		LAKE MARY, FL	DOWNTOWN PITTSBURGH, PA

Variations

As described above, the scenario involves submitting tracking numbers in batches. As a result, performing Action 1.3.1 requires matching each piece of status information with the corresponding tracking number, based on the value in the left hand column of the result. A likely variation would involve submitting the tracking numbers individually, which would be simpler but probably slower.

Instead of storing only DHL numbers, the spreadsheet might contain a list of shipments carried by different shippers. In that case, the scenario would need to choose the right web site to look up the status information, conditional on the format of each tracking number.

Another set of variations relate to the fact that when accessing the DHL site for the first time, the user must select a country (such as USA). At that point, the site forwards the browser to the page shown in Action 1.2. It also sets a cookie so that the site can automatically forward the browser on future visits.

- In one variation, the user has already set the country before demonstrating the scenario to a macro recorder. If this cookie already exists before the user records the macro, and then the user wants to share the macro or use it on another machine, then the macro will not contain any instructions for selecting a country. In that case, it may fail when replayed.
- On the other hand, if the user records the cookie during the macro, then the macro will contain a step for selecting the country. However, the cookie will probably already be present when the user replays the macro. In that case, when the user replays the macro, the site will automatically forward the browser to the page shown in Action 1.2, so the macro will not have an opportunity to replay the step for selecting a country. Thus, the macro may fail when replayed.

In either situation, a naïve implementation of a macro tool may fail. One possible approach to coping with this would be to turn the browser's cookie management off for this site, so that the "select country" step always occurs. This would permit automating the scenario.

Macro Maintenance

According to the Internet Archive's Wayback Machine, the form in Action 1.2 has changed slightly since it appeared in April 2004. Although the main text field's name has not changed, two hidden fields ("hdnPostType" and "hdnRefPage") have been removed, while one new hidden field ("nav") has been added. In addition, a client-side validation script has been replaced with equivalent server-side code. It is possible that page evolution of this sort would require some macro maintenance.

The colors and font of the web form have not changed significantly since April 2004, but the page layout has. Although the Wayback Machine does not contain any archived versions of the output page (Action 1.3), it seems possible that the layout of this output page stayed consistent with the style of the main web form. Evolution of this sort might prompt macro maintenance activities.

Scenario Source

Several of the authors have needed to track multiple shipments at Christmas, though none of us has actually used a spreadsheet to do so. We hypothesize that reading the information from a spreadsheet (as described in the scenario above) might provide an easy-to-use interface for automating the process.

Path to Procurement — Typical User: office worker

Overview

When office workers at IBM want to buy office supplies, they must use a certain intranet application. Unfortunately, the application is a labyrinth of confusing links and forms, which makes it difficult for workers to locate the forms that they need to order items.

It is particularly challenging to find the form for ordering items that do not appear in the online catalog. This scenario describes the steps required to find this specific form and fill it out.

Starting Conditions

The user has a spreadsheet listing items to order. Each row contains the following information:

- Item description
- Item quantity
- An estimated unit price for the item

Result

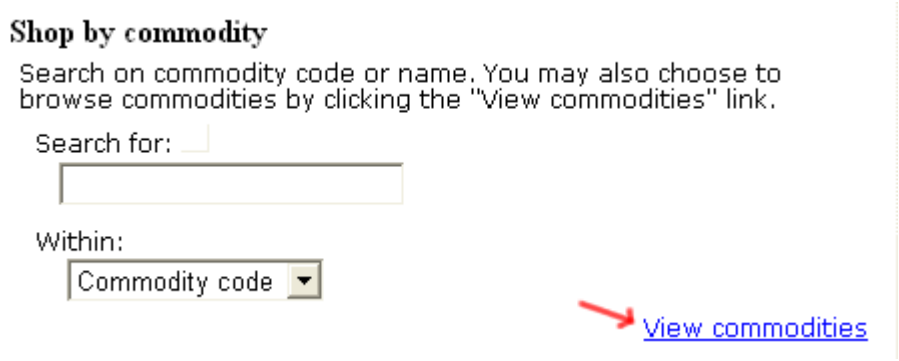
The user's shopping cart contains the desired items.

Actions

1. Open the procurement web site.
2. If the server sends a login page, then
   2.1. Submit the username and password.
   2.2. Attempt again to open the procurement web site.
3. Follow the "View Commodities" link.
4. Follow the "MRO/SUPPLIES" link.
5. Follow the "GENERAL OFFICE SUPPLIES" link.
6. Follow the "Continue" link.
7. Follow the "All other orders" link.
8. For each item to be ordered,
   8.1. Enter the description, quantity, and estimated price.
   8.2. If this is the last item, then
      8.2.1. Follow the "Add and go to cart" link.
   8.3. Else
      8.3.1. Follow the "Add and stay here" link.

Action Details

3 The scenario begins by clicking a long series of links, beginning with "View Commodities".



4

W	LOGISTICS	Payments and costs associated with logistics ser
K	MANUFACTURING EQUIPMENT	All services and supplies associated with the ren equipment.
R	MRO / SUPPLIES	Operating supplies which include Maintenance, supplies, business cards, books/subscriptions and
T3	PASSTHRU PAYMENTS	Direct cost reimbursement for payments made t

5

W79	NON-OEM REPAIR - OFF SITE	manufacturing tool repair, use S80 under Family
W80	NON-OEM REPAIR - ON SITE	Non OEM repair On-site - Repair of IBM equipn manufacturing tool repair, use S80 under Family
W14	GENERAL OFFICE SUPPLIES	General office supplies, i.e. pens, notebooks, tap
W13	OFFICE SUPPLIES CONTAINING	Office supply chemicals such as adhesives, aeros disinfectants, dusters, epoxies, glue, glue sticks, i

6

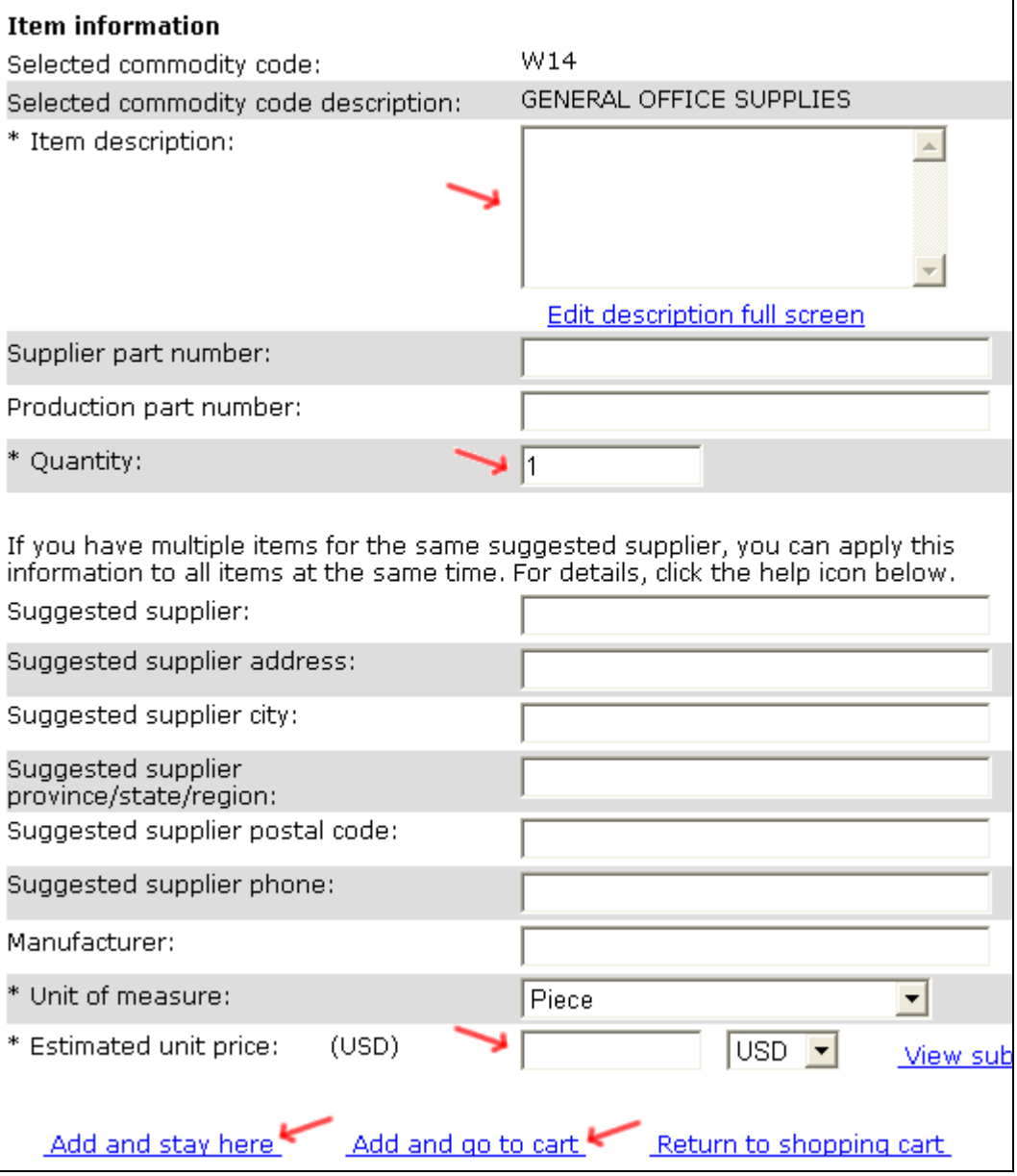
Return to previous page Continue

7

Subcommodities

Preferred shopping methods	Supplier	Special instru
Staples - NON Catalogs Orders	STAPLES NATIONAL ADVANTAGE	Only use this op
Staples office supplies catalog (non chemical)	STAPLES NATIONAL ADVANTAGE	Please ensure yc
All other orders		Use this option available from t

8 For each item to be ordered, enter the description, quantity, and estimated price.
For the last item, use the "Add and go to cart" link; otherwise, use the "Add and stay here" link.



Variations

Variations might include entering information into optional fields of the form, such as "suggested supplier" and "suggested supplier address," or selecting an alternate "unit of measure."

Other variations relate to the fact that the user must log into the application. If the user has logged in before recording a macro, but has not logged in before replaying the macro, then the macro tool will need to deal with the authentication page. Conversely, if the user has not logged in before recording the macro, then the recorded macro will contain the username and password; in this case, the macro tool should support encryption of the password as well as parameterization of the password if the tool supports sharing of macros among users.

Note that the scenario modifies the user's shopping cart. Consequently, if a macro for this scenario failed partway through the scenario, then it would be undesirable to restart the macro from the beginning, since that would resulting in adding duplicate entries to the shopping cart. In that case, a variation of this scenario would take place, in which the scenario begins where the previous one left off.

Macro Maintenance

The application is on an intranet, so the Internet Archive's Wayback Machine does not contain any archived versions of this application. However, it is still possible to discern some evolution in the application.

For example, the source code to Action 6 contains an HTML comment indicating that a new hidden field was added to an invisible form in July 2005. The "Continue" link fires a JavaScript function that submits this form. Consequently, if a user recorded a macro prior to that date, and the macro tool only replayed http requests (rather than actually clicking on the link using DHTML events) then the macro tool would fail to include this new hidden field in its http request. As a result, the application might not work properly. This could prompt macro maintenance activities.

Scenario Source

A co-worker at IBM sent an email describing this scenario. Right now, co-workers manually type item information into the web application, but we hypothesize that reading the information from a spreadsheet (as described in the scenario above) might provide an easy-to-use interface for automating the process.

Peoplesoft Scraper — Typical User: IT staff

Overview
Information technology staff sometimes must export a list of employees from a human resources (HR) system, such as Peoplesoft, which has a web form interface. This is useful when loading the information into another database or application.

In this scenario, the staff must export a list of employees under a certain manager.

Starting Conditions
The Peoplesoft web application provides a search form for retrieving lists of employees. The list spans multiple pages, with a link to go to the next page. For each employee, there is also a link to a page that displays the employee's detailed information.

The user wants to retrieve a list of employees in a certain department.

Result
For each employee, the spreadsheet contains a row with the following employee information:

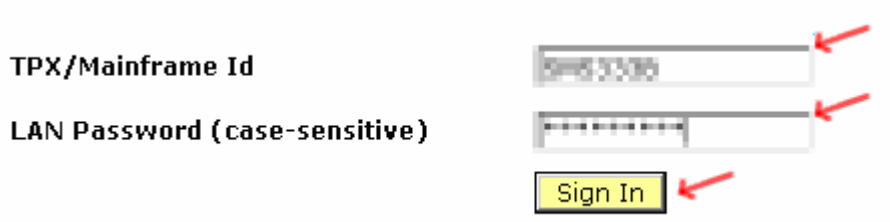
- Name
- Phone number
- Office code
- Job Title

Actions
1. Follow the "HR Express" link.
2. Log into the Peoplesoft web application.
3. Submit the manager name.
4. For each page of the search results,
   4.1. For each employee listed on this page of results,
      4.1.1. Follow the link to the page showing the employee's detailed information.
      4.1.2. Retrieve the employee's name, phone number, office code, and job title.
   4.2. Back on the list of search results, click on the link to the next page of results.

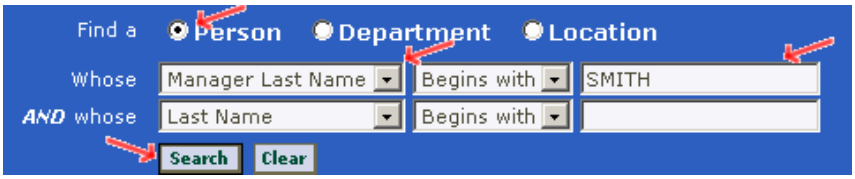
Action Details

1 Follow the "HR Express" link.

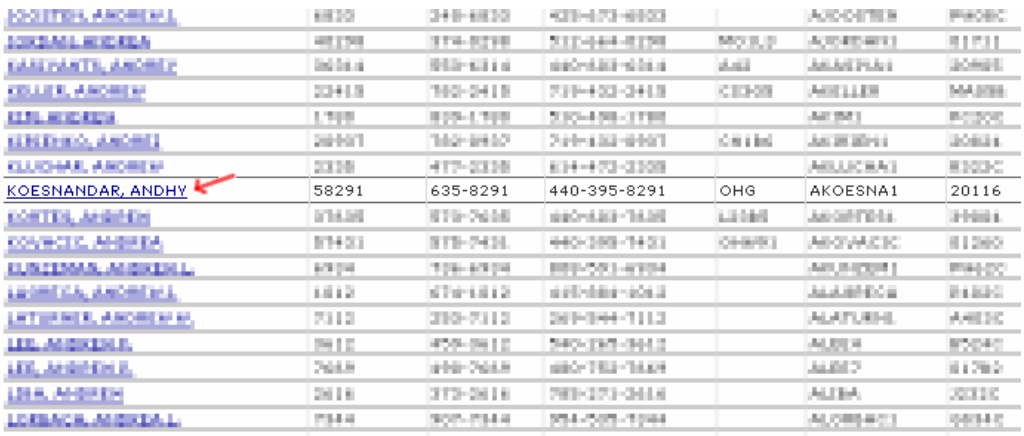
2 Log into the Peoplesoft web application.



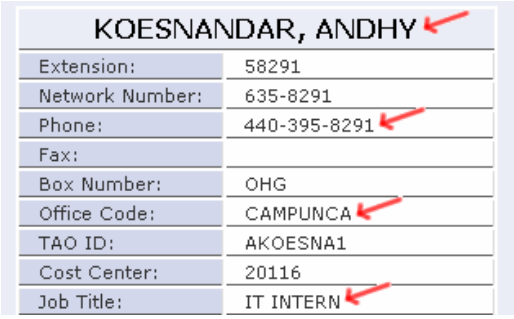
3 Fill out the form to search for persons by manager name.



4 For each employee on each page of results, click on the employee's name to open a popup window containing the employee's detailed information.



4.1 Retrieve the employee's name, phone number, office code, and job title.



Variations

The IT staff may want to search for employees based on department, cost center, job title, or similar filters provided by the Peoplesoft search form. To parameterize these variations, a macro for the scenario above might actually start with Action 4, thereby allowing the user to specify any search that the Peoplesoft system supports.

Unlike the users in most of these scenarios, the members of IT performing this scenario are quite skilled. They are probably capable of writing regular expressions, manipulating databases through SQL, and writing batch files. Such users will be able to take advantage of more complex features offered by any macro recorder. For example, the IT staff may want to filter employees based on a regular expression over fields of the results. Possibilities include filtering out employees whose date of hire is prior to the past month. To facilitate such variations, the macro tool could support regular expression filters.

After exporting the records to Excel, it is generally necessary to import those records into another system, such as another site's Peoplesoft installation. In that case, users will need to create a macro that posts each row of the spreadsheet to another web form. It is conceivable that a single macro could perform the export and import operations.

Macro Maintenance

Because of the numerous possible scenario variations listed above, it seems likely that the IT staff will want to reuse, adapt, and tweak macros for this scenario.

Indeed, research has revealed "that system administrators often need to build small custom tools quickly as they configure, monitor, or troubleshoot systems end-to-end" [6]. In many cases, because of similarities among different web applications monitored by these custom tools, the staff may be able to create a new tool by modifying an existing tool.

One traditional approach to facilitating reusability of code is to provide mechanisms for information hiding within libraries and components. In contrast, for IT staff, maintaining a repository of tweak-able custom tools seems to be the most feasible approach for facilitating reusability [6].

Scenario Source

One of the authors has observed the IT staff performing this scenario at an organization where he worked.

Per Diem Lookup — Typical User: office worker

Overview
To file travel expense reports, office workers use an intranet application with fields for the date and locality of the travel (city and state). Using the date and locality, they must enter the federally-approved per diem allowance into another field.

They generally achieve this by popping open a new window (leaving the expense report form running in a window in the background), then navigating to a government web site to look up per diem rates. After using the site to look up the rate for that date and locality, workers copy-paste the result back into the expense report and close the popup window.

Starting Conditions
The user already has open an expense report web form containing the following data in text widgets:

- A date, in `MM/DD/YYYY` format
- A locality, in `City, ST` format

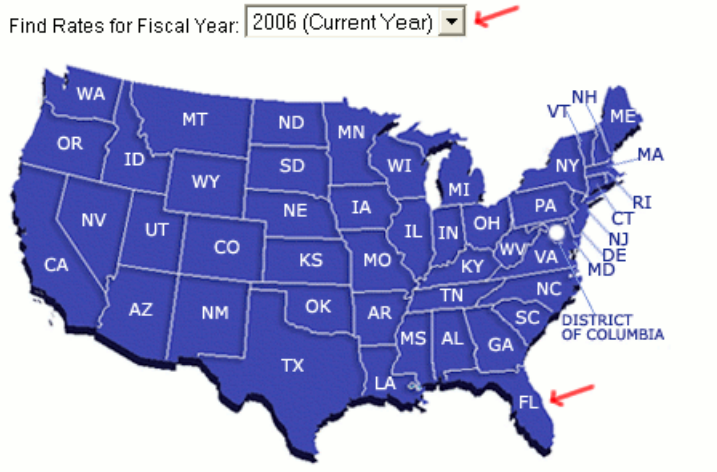
Result
The clipboard contains the per diem rate for the date/locality.

Actions
1. Open the per diem web site.
2. Select the year and click the image map link for the state.
3. If the desired city and date appear, then
   3.1. Retrieve the per diem rate and terminate.
4. Open the county lookup web site.
5. Submit the city name.
6. If the city's county appears, then
   6.1. If the county also appears on the per diem web site, then
      6.1.1. Retrieve the per diem rate and terminate.
7. Use the default value from the per diem web site (if Action 6 does not generate a rate).

Action Details

1 The site is publicly accessible at
`http://www.gsa.gov/Portal/gsa/ep/contentView.do?programId=9704&channelId=-15943&ooid=16365&contentId=17943&pageTypeId=8203&contentType=GSA_BASIC&programPage=%2Fep%2Fprogram%2FgsaBasic.jsp&P=MTT`

2 Select the year and click the image map link for the state.



The map has an image map; each link has an href like `javascript:setAction('Florida')`

3 Look for the appropriate per diem rate based on the city and the travel date. If the desired city and date appear, then retrieve the per diem rate and terminate. Otherwise, proceed to Action 4 to try looking up a value based on county name.

Primary Destination (1)	County (2, 3)	Max Lodging (exc. taxes)	+	M&IE Rate	=	Max Per Diem Rate
ALTAMONTE SPRINGS	SEMINOLE	67		43		110
BRADENTON	MANATEE	64		35		99
COCOA BEACH (October 1 - December 26)	BREVARD	83		39		122
COCOA BEACH (December 27 - September 30)	BREVARD	93		39		132
DAYTONA BEACH (February 1 - March 31)	VOLUSIA	134		43		177
DAYTONA BEACH (April 1 - January 31)	VOLUSIA	79		43		122

The column labels are TD tags containing IMG tags with `alt="Primary Destination"` and `alt="Max Per Diem Rate"`. A BR tag separates each city name from the date range.

4 Sometimes the user can find a per diem rate for the county, even when the specific city is unavailable. However, the user rarely knows the county corresponding to the city. Fortunately, the per diem web site has a link to the "NACO" site, which helps users find the city's county.

Cities not appearing below may be located within a county for which rates are listed. To determine what county a city is located in, click here for the National Association of Counties (NACO) website (a non-federal website).

5 The NACO web site offers several ways to look up a city's county. Perhaps the easiest is to type the city's name into the fourth form on the web page, shown below.

Search By Name of City

Enter the name of a city to find the county in which the city resides.

City Name: Rockledge [Search for County]

6.1 See if the city (with the correct state) appears on the NACO results. If so, then retrieve the county name and proceed to Action 6.2. Otherwise, go to Action 7 to compute a default per diem rate.

State	Place			County
AL	Rockledge			Etowah County
FL	Rockledge	city	Incorporated Area	Brevard County
GA	Rockledge			Laurens County
PA	Rockledge	borough	Incorporated Area	Montgomery County

6.1.1 If the county name was available, then the user can try to look up a per diem rate based on the travel date and county. If this is successful, then the scenario terminates. Otherwise, the user proceeds to Action 7.

Primary Destination (1)	County (2, 3)	Max Lodging (exc. taxes)	+	M&IE Rate	=	Max Per Diem Rate
ALTAMONTE SPRINGS	SEMINOLE	67		43		110
BRADENTON	MANATEE	64		35		99
COCOA BEACH (October 1 - December 26)	BREVARD	83		39		122
COCOA BEACH (December 27 - September 30)	BREVARD	93		39		132
DAYTONA BEACH (February 1 - March 31)	VOLUSIA	134		43		177
DAYTONA BEACH (April 1 - January 31)	VOLUSIA	79		43		122

7 The government site specifies a per diem rate to use as a default when the Actions above fail to generate a result. This appears in a grey background text box near the top of the page.

NOTE: If neither the city nor the county is listed, the location is a standard CONUS destination with a rate of $60.00 for lodging and $31.00 for meals and incidental expenses (M&IE).

State Tax Rates & Exemption Forms

Properties at Per Diem (FedRooms)

Select another State

Primary Destination (1)	County (2, 3)	Max Lodging (exc. taxes)	+	M&IE Rate	=	Max Per Diem Rate (4)	First & Last D (75% of M
ALTAMONTE SPRINGS	SEMINOLE	67		43		110	32.25
BRADENTON	MANATEE	64		35		99	26.25
COCOA BEACH (October 1 - December 26)	BREVARD	83		39		122	29.25
COCOA BEACH (December 27 - September 30)	BREVARD	93		39		132	29.25

The default per diem rate actually is the sum of two numbers ($66.00 and $31.00 in the screenshot). These values vary from year to year, depending on what year selected in Action 2. Thus, the default per diem is not a constant, and determining the right value involves picking these two currency amounts out of the text and summing them.

Variations

Automating this with a macro might copy this value directly back to a widget in the expense report form rather than leaving it in the clipboard.

If the city is located outside the United States, then these actions will not return a result. We do not know how users choose the right per diem rate in this case, so the macro player might show an alert and let the user take alternate actions.

Macro Maintenance

Although the government currently publicizes per diem rates at the URL shown above, they previously appeared at `http://policyworks.gov/org/main/mt/homepage/mtt/perdiem/travel.htm`

The HTML for the current table of rates actually contains hidden (commented) code for an additional column called "Properties at Per Diem". If it was not commented out, then it would appear on the right side of the table and would contain a link to a list of hotels that honor the per diem rates. It appears that this column used to be visible but has since been commented out. Although this would not affect any macro automating the scenario above, it could affect any macro that attempted to operate on this list of hotels.

The government occasionally posts alerts on the per diem web site, and these may force the user to make modifications. For example, since we first documented this scenario, the government has posted a document with special rules for traveling to areas affected by Hurricane Katrina. The user might want to add special rules to the macro to handle these new government instructions.

The Internet Archive's Wayback Machine has an archive of the NACO county lookup site from April 2003. This reveals that the city widget's name has not changed; however, the text on the submit button (which has no name) did change, from "Search for City" to "Search for County." Any macro that targeted this button based on human-readable label would have required maintenance activities.

In addition, the current version of this page provides four different ways of looking up the county, whereas the old version only provided two ways. Although the output page (Action 6.1) is not available in the Wayback Machine, it seems possible that the addition of new functionality may have resulted in modifications to the output page. This could have prompted maintenance activities.

Scenario Source

Several administrative assistants performed this scenario during our contextual inquiry.

Person Locator Scraper — Typical User: volunteer developer

Overview
After Hurricane Katrina, dozens of people created "person locator" sites to help end users search for survivors [20]. Unfortunately, these sites were largely redundant, forcing end users to visit numerous sites. Consequently, a few teams aggregated these dozens of sites into large databases so that users could search through the data in less time.

To aggregate the existing sites, these teams created custom screen scrapers in PHP, Perl, Ruby, and similar languages to retrieve information from existing sites and store it in an XML document. The scraper's author could then upload the XML document to a server that appended the records to the central database.

Because the scraped site contains thousands of records, it is extremely inefficient to repeatedly scrape records that have already been scraped before. In addition, repetitively scraping these records and adding them to the central database would result in needless duplicates in the central database. Consequently, this scenario's screen scraper contains actions aimed at avoiding repetitive downloads of records.

The scenario below describes the steps that one PHP scraper performed.

Starting Conditions
A page on the existing web site currently displays a list of people and their status. This list spans hundreds of pages, with each page containing a few dozen records (one record per lost person). Pages are sorted from oldest to newest, and within each page, records are sorted from oldest to newest.

Result
The output XML document contains one record for each person.

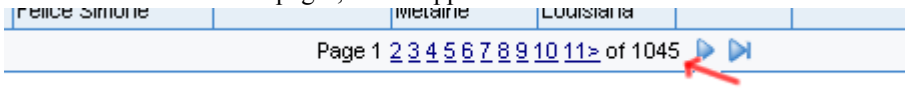
The scraper writes a timestamp of the current date/time in a file. That way, the next time it executes, it can examine this "last execution" timestamp and skip past pages containing records that it has already retrieved.

Actions
1. Open the source web site.
2. Read the total number of pages.
3. If there is no "last execution" timestamp set, then
   3.1. Make a note that the "start page" below is 1.
4. Else
   4.1. Loop backward from the last page to the first page.
      4.1.1. If the current page contains data prior to the "last execution" timestamp, then
         4.1.1.1. Make a note that this is the "start page" below, and exit the loop.
5. Write a timestamp for the current date/time to a file.
6. Beginning with the "start page", for each page of records,
   6.1. For each person record,
      6.1.1. Parse the record's fields and append the record to an array in memory.
      6.1.2. Augment the record with information about the record's author.
7. For each record in memory,
   7.1. Append to XML.

Action Details

1	Open the source web site, which is publicly accessible at `http://www.publicpeoplelocator.com/index.php?peopleOrder=Sorter1`

2 Read the total number of pages, which appears at the bottom of the screen.



The scraper uses the regular expression ` of (\d+) ` to retrieve this value.

3 The scraper uses a file to store the timestamp when the scraper last executed.

If the scraper never executed before, then this file is empty. In this case, the scraper needs to read all pages of data. So it sets the "start page" variable to 1, then skips to Action 5.

If the scraper has executed before, then this file contains a "last execution" timestamp. In this case, the scraper must determine the page where the new data begin. So it proceeds with Action 4.1 to determine the correct "start page."

4.1 If the scraper executed before, then it must determine where the pages of new data start.

Recall that pages contain data in sorted order. Pages with the highest numbers contain the most recent data. So to determine where the pages of new data start, the scraper loops backward from the last page toward the first page, looking for a page that contains data older than the current timestamp.

Retrieving a page of data involves appending the page number to the following URL:
`http://www.publicpeoplelocator.com/index.php?peopleOrder=Sorter1&peoplePage=`

4.1.1 Each record has a date/time in the rightmost column. The scraper reads the date/time of the first row of the page. If that date/time is prior to the "last execution" timestamp, then it assumes that this page contains a mixture of old and new data. So this page will serve as the "start page" in Action 6 below.

State	Zip	Telephone	Status	Last Update Date
Louisiana			Looking For	6/25/2006 10:32:01 AM
Louisiana	70458		Safe	6/26/2006 3:13:38 AM
Louisiana			Hospitalized	6/26/2006 11:07:23 AM
Louisiana				6/26/2006 5:54:03 PM
Louisiana	70117			6/26/2006 5:57:10 PM
				6/26/2006 6:01:04 PM
Louisiana			Safe	6/27/2006 4:37:49 AM

The scraper retrieves the date/time by looking for the ninth column (which happens to be the last column) and extracting the value using the following regular expression:

```
/person_update.php?[^s]+son_id=(\d+)">([^<]*)<\\/a> <\\/td>[^>]+>
(.*) <\\/td>[^>]+>(.*) <\\/td>[^>]+>(.*) <\\/td>[^>]+>
(.*) <\\/td>[^>]+>(.*) \\/td[^>]+>(.*) <\\/td>[^>]+>
(.*) <\\/td>[^>]+>([^<]*)
```

5 The scraper writes a timestamp for the current date/time to a file. When the scraper executes in the future, this will be the "last execution" timestamp.

6 Having determined the "start page," the scraper now proceeds with the primary work of scraping records from the web site. Beginning with the start page and going through to the last page, it retrieves each page by appending the page number to the following URL:
`http://www.publicpeoplelocator.com/index.php?peopleOrder=Sorter1&peoplePage=`

6.1.1 Each retrieved page contains a few dozen records. The scraper uses the same lengthy regular expression shown in Action 4.1.1 to break up the row into pieces that it assigns to array slots in memory:

- First Name
- Last Name
- Home Street
- Home City
- Home State
- Home Zip
- Phone Number
- Status Description
- Source Date

In the process of copying record fields to memory, the scraper unescapes HTML character codes (for example, replacing "&" in the HTML with its "&" equivalent).

In addition, it sets a Boolean for each record to represent if the person is found. This Boolean is set to true only if the person's status is "Safe", "Moved/Moving", "Hospitalized", or "Deceased".

6.1.2 The first column of the record table contains a link to an "update" page that tells about the record's author (and provides a form for updating the record, thereby resetting its date/time to the current date/time). The scraper downloads the HTML that this link points to and parses out the record's most recent author, as well as any messages from the author.

7 Finally, for each person record in memory, the scraper writes the person's data to an XML file. The specification for this file is publicly available at `http://zesty.ca/pfif/1.1/`

The XML document contains a list of person tags. The subtags within each person tag contain fields of the person record. The scraper must perform a substantial amount of reformatting when copying these fields into the XML tags (in addition to the usual XML escaping rules):

- The scraper reformats dates to `Y-m-d\TH:i:s\Z` format.
- The scraper removes accents, umlauts, and so forth (essentially transcribing from latin-1 encodings to lower-ASCII near-equivalents).
- The scraper converts all state names to two-letter abbreviations.

Variations

Some other Hurricane Katrina sites require users to authenticate in order to read the site. Scrapers for those sites would need to perform a login Action.

The scraper's source code suggests that it is compatible with a scenario variation in which the script is executed multiple times. In this case, Action 4 carefully tries to avoid repeatedly copying every record each time. Although this probably helps to improve performance, it does not completely eliminate copying records multiple times. (It would be impossible to avoid duplicates, anyway, since duplicates may exist on other sites downloaded by other scraper teams.)

Macro Maintenance

We suspect that the scraper as originally coded did not attempt to avoid duplicate work by looking backward through the pages to find the start of the new data. The reason for suspecting this is that the regular expression used in Action 4.1.1 (to retrieve the date/time) is the same as the regular expression used

in Action 6.1.1 (to parse the records), even though a far simpler regular expression would have sufficed for Action 4.1.1. It seems as though the programmer coded this powerful regular expression in order to use it in Action 6.1.1 and only later perceived its usefulness in Action 4.1.1 while adding the code for avoiding duplicate work.

It also appears that the programmer had attempted to serialize the array of person records to a file, and then unserialize it upon the next scraper execution. We suspect this because the scraper contains code for serializing the array, but the corresponding unserialization code is commented out.

This scraper contains no error handlers. For example, it cannot deal with the possibility that the site might evolve in a way that breaks the regular expression that retrieves the page number in Action 2. Therefore, possible maintenance might include adding error handlers as well as checks to determine if the site is evolving. For example, if the site's owners inserted a new column into the middle of the table of results, then this scraper probably would not notice; it would probably continue to execute as usual, but it would match fields to the wrong locations in the XML output. A more robust version of this scraper might parse the records based on column headers rather than simple column position.

The code for this scraper is publicly available at
`http://lardbucket.org/projects/katrina/scrape_ppl.phps`

Scenario Source

We discovered this scraper while preparing for a series of interviews with creators of Hurricane Katrina person locator web sites [20].

Scraper for CMS — Typical User: webmaster

Overview
One of a webmaster's repetitive tasks is scraping data off one site and copying it into another site's content management system (CMS). Typically, the interface to the target CMS is a web form, as in the scenario below, though the interface is sometimes a text file.

Starting Conditions
The source web site, called THEHFA, contains information for a list of events. This list spans multiple pages, with a "Next" link in the source window to access successive pages.

In addition, the user has logged into the CMS for the target web site, called AHFA.

Result
The fields of the event information have been copied into the AHFA CMS.

Actions
1. Direct the browser windows to the correct pages.
   1.1. Open the THEHFA events page in the source window.
   1.2. Open the AHFA events CMS in the target window.
2. For each page of events in the source window,
   2.1. For each event in the source window,
      2.1.1. Scrape the event name, link, email, date, location, description, and image.
      2.1.2. Submit this event data in the target window.
   2.2. Click the "Next" link to access the next page of events.

Action Details

1.1 The THEHFA web site is publicly accessible at `http://thehfa.org/`

There, click on the "Events" link on the left hand side of the page.

The href attribute of the "Events" link is dynamically generated; it differs each time a new user visits the page. Specifically, the href always begins with `http://www.thehfa.org/Events.cfm?`, and then a pseudo-random CFID and CFTOKEN parameter is appended to the URL. (This is how some sites manage user sessions without using cookies.) The page apparently works fine if the CFID and CFTOKEN parameters are omitted.

1.2 No screenshot for the AHFA events CMS is available. However, we recall that the page had a form with the following six textboxes:

- Event Name
- Event URL
- Event Date

- Event Banner
- Email
- Description (a multi-line textbox)

2.1 For each event in the source window, submit the six pieces of event details through the target window.

- A single TD tag contains all six of the event's fields. Within that TD, different HTML tags bracket each event field.
- The event title is contained in a B tag nested inside of two FONT tags.
- The event URL is contained in an A tag and is separated from the preceding line by a BR tag.
- The event contact info also is contained in an A tag.
- The event date is contained in a FONT tag.
- The event banner is contained in a FONT tag; a BR tag separates it from the event date.
- The event description is contained in a P tag.

2.2 If a "Next" link appears, then click it to access the next page of events.

The link to access the next page looks like a circle with an arrow inside of it. It appears near the top right of the screen, and then again near the bottom right of the screen. The link is preceded by links that enable the user to skip ahead to other pages.

The URL of this link is dynamically generated so that it always takes the user to the page immediately after the current page. If there is no "next page," then this link does not appear. The link contains an IMG tag with `alt="Next Page"` and an href that contains "next.gif".

Variations

The target AHFA events CMS uses text fields and an image upload button, but more complex CMS web forms may provide sophisticated HTML-editing widgets rather than simple text fields. For example, Ektron is a vendor that sells an HTML-editing widget called eWebEditPro. If the CMS provides an HTML-editing widget like this, then the macro tool will encounter much more complexity while automating the scenario, since it will need to fire DHTML events in order to insert the data (rather than simply setting the value attribute of a textbox).

Other variations will arise based on whether the user intends to copy all or some of the events data into the target CMS. Presumably any events already in the CMS should not be copied a second time into the CMS the next time that the user performs the scenario. Thus, the user will probably want to set some conditional stopping condition for any macro automating this scenario. Most CMS's provide a page that lists all the records already within the system. However, the layout of this page varies by CMS. Therefore, the means for testing a stopping condition will also vary by CMS.

Since this scenario submits information through the CMS, the target server's state changes. Consequently, if the user recorded a macro for this scenario, and the macro failed halfway through, then re-starting the macro from the beginning might result in duplicate entries on the target site (depending, of course, on the details of the macro and the CMS). Therefore, during a re-start, a scenario variation would take place, in which the actions begin where the previous scenario left off.

Macro Maintenance

The Internet Archive's Wayback Machine contains entries for `http://www.thehfa.org/Events.cfm` from as far back as July 2001. Unlike the current version, the older version did not have a contact email address for each event. When THEHFA added this email address, the new line of data might have thrown off any macros recorded prior to that point, causing the macro tool to put the new email address into the date field. This could have prompted maintenance activity.

Scenario Source

During a previous job as a webmaster, one of the authors used to maintain the AHFA site using the CMS involved in this scenario.

Staff Lookup — Typical User: office worker

Overview
Office workers sometimes create a spreadsheet listing several co-workers and their contact information. For example, collecting this contact information into one document may be necessary before sending it off to another person. Alternatively, as we observed in this scenario, the document might be printed and handed out during a meeting or used for managing a conference attendee list.

To collect this information into one spreadsheet, the worker must copy each co-worker's name into a web form, submit the form, and then copy fields from the result back into the spreadsheet, as demonstrated by this scenario.

Starting Conditions
A spreadsheet contains a list of co-worker's names, one per row.

Result
The spreadsheet contains several additional fields for each co-worker:

- Title
- Phone number
- Email address

Actions
1. Open the staff directory web site.
2. For each co-worker,
   2.1. Submit the co-worker's name.
   2.2. If the server returns zero records, then
      2.2.1. Terminate with error condition.
   2.3. If the server returns multiple records, then
      2.3.1. Follow the link for one of the records (so the server shows just that one record).
   2.4. Retrieve the title, phone number, and email address.

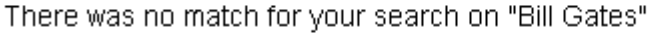
Action Details

1 The staff directory web site is publicly accessible at `http://people.cs.cmu.edu/`

2.1 Submit the co-worker's name.

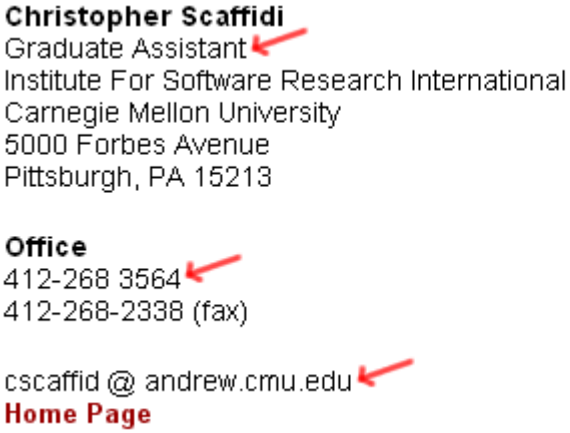
Since the textbox has no button to submit the form, the user hits an "Enter" keystroke.

2.2 If the server returns zero results (just an error message like the one below), then terminate with error condition.



2.3 If the person's name is very common (for example, entered as just "Christopher"), then server may return multiple results. In order to choose a record, the user must rely on extra knowledge, such as the person's department, which was *not* encoded in the input spreadsheet alongside the person's name. Clicking on a record brings up the detail page shown in Action 2.4.

2.4 Retrieve the title, phone number, and email address.



Variations

When the server returns more than one result, the user must rely on extra knowledge such as co-workers' department abbreviations in order to pick the right row. If a macro tool lacks this information, then it cannot reliably pick the right entry. Therefore, a likely variation of this scenario would simply report an error condition if the directory returned more than one result.

Another likely variation is to retrieve the information for every person in the database. This list can be retrieved via a specific URL (`http://people.cs.cmu.edu/a_z/ALL.html`), and the web page looks similar to the screenshot shown in 2.3.

At some organizations, accessing the staff directory may require authentication.

Note that the phone number and email address are malformed—they each contain spaces in odd places. In a likely scenario variation, the user or a macro might clean these up by replacing the phone number's spurious space with a hyphen and stripping out email address's spurious spaces. (It is conceivable that programmers have intentionally inserted these spaces to throw off spammers' email harvesters.)

Macro Maintenance

The Internet Archive's Wayback Machine contains records back to June 2001 for the search form and search results. There have been no changes to the widgets' names, nor to the appearance of the results. (In fact, even the malformed formatting of the phone number and email addresses has remained constant.)

It seems likely that if a user recorded a macro for this scenario, that it would be desirable some day to extend the macro to retrieve the other fields of the record.

Scenario Source

One manager performed this scenario during our contextual inquiry of office workers.

Stock Analysis — Typical User: financial analyst

Overview
Some investors perform complex analyses on stock data before choosing where to invest their money. These analyses often use information that is freely available, but it is scattered across several web sites. Thus, investors must perform a tedious series of actions to retrieve this information into a spreadsheet. The spreadsheet can process this information to compute various statistics.

Starting Conditions
The user has a spreadsheet with several rows. Each row has a stock symbol s_i in the first column and a corresponding date d_i in the second column.

Result
In addition to s_i and d_i, each spreadsheet row contains new information downloaded from web sites. The information includes the following:

- The High, Low, and Closing prices for s_i, as well as the daily Volume, for the 200 days prior to d_i.
- The Price-to-Earnings and Price-to-Sales ratio for s_i in the year d_i.

Actions
1. For each ticker symbol,
   1.1. Open the Yahoo! Finance web site.
   1.2. Submit the ticker symbol.
   1.3. Follow the Historical Prices link.
   1.4. For each page of results,
      1.4.1. For each date in the results,
         1.4.1.1. Retrieve the corresponding data items.
   1.5. Open the MSN Finance web site.
   1.6. Submit the ticker symbol.
   1.7. If the desired date appears in the table, then
      1.7.1. Retrieve the corresponding data items.
   1.8. Else
      1.8.1. Follow the Price Ratios link.
      1.8.2. Retrieve the desired data items.

Action Details

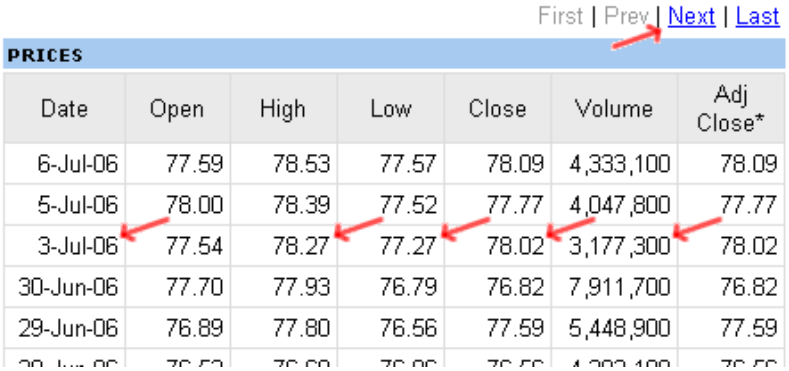
1.1 The Yahoo! Finance web site is publicly accessible at `http://finance.yahoo.com/`

1.2 Submit the ticker symbol.

1.3 Follow the Historical Prices link.

1.4 Yahoo! returns a list of the stock's data, with one record for each trading day. These records span multiple pages that the user can access via the "Next" link.

Some of these records might be for days after d_i. The user retrieves all 200 pieces of data prior to the specified date d_i and stores them in row i of the spreadsheet. For each data point, the user retrieves the High, Low, Close, and Volume values.



First | Prev | Next | Last

PRICES

Date	Open	High	Low	Close	Volume	Adj Close*
6-Jul-06	77.59	78.53	77.57	78.09	4,333,100	78.09
5-Jul-06	78.00	78.39	77.52	77.77	4,047,800	77.77
3-Jul-06	77.54	78.27	77.27	78.02	3,177,300	78.02
30-Jun-06	77.70	77.93	76.79	76.82	7,911,700	76.82
29-Jun-06	76.89	77.80	76.56	77.59	5,448,900	77.59

1.5 The specific MSN Finance web page required for this scenario is publicly accessible at `moneycentral.msn.com/investor/invsub/results/compare.asp?Page=TenYearSummary`

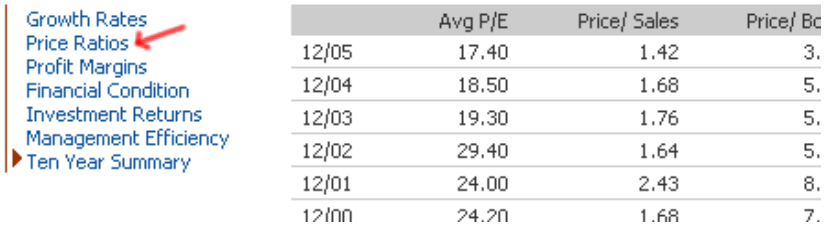
1.6 Submit the ticker symbol.

Name or Symbol: IBM Go Find Symbol Print Report

1.7 MSN returns a table listing the stock's data for each of the past ten years. The user picks the row of this table that corresponds to the year of interest (actually, the December of the year that d_i is in), and then retrieves the Average P/E and Price/Sales statistics for that row.

	Avg P/E	Price/ Sales	Price/ Book	Net Profit Margin (%)
12/05	17.40	1.42	3.91	8.8
12/04	18.50	1.68	5.45	8.8
12/03	19.30	1.76	5.64	8.5
12/02	29.40	1.64	5.86	6.6
12/01	24.00	2.43	8.83	9.0
12/00	24.20	1.68	7.27	9.2
12/99	26.70	2.20	9.50	8.8
12/98	21.70	2.07	8.80	7.7

1.8.1 If the date does not appear because d_i falls in a year that is incomplete (e.g.: any date in 2006 for these screenshots), then the user clicks on the Price Ratios link to get the current year's data.



Growth Rates
Price Ratios
Profit Margins
Financial Condition
Investment Returns
Management Efficiency
Ten Year Summary

	Avg P/E	Price/ Sales	Price/ Bc
12/05	17.40	1.42	3.
12/04	18.50	1.68	5.
12/03	19.30	1.76	5.
12/02	29.40	1.64	5.
12/01	24.00	2.43	8.
12/00	24.20	1.68	7.

1.8.2 The user retrieves the Current P/E Ratio and the Price / Sales Ratio for the stock.

Price Ratios	Company	Industry	S&P 500
Current P/E Ratio	15.2	21.0	17.6
P/E Ratio 5-Year High	61.4	184.5	64.8
P/E Ratio 5-Year Low	14.7	17.4	16.6
Price/Sales Ratio	1.36	1.23	1.46
Price/Book Value	3.70	3.02	2.76
Price/Cash Flow Ratio	9.00	11.80	11.90

Variations

As noted earlier, stock analyses can be rather complex and probably represent requirements for the target spreadsheet rather than requirements for the macro recorder/player. Consequently, we omitted the details of these computations from the main scenario Actions above.

Nonetheless, the user who contributed this scenario wanted to perform some of these calculations within the macro and write final values back to the spreadsheet (rather than writing intermediate values and using the spreadsheet to compute final values). For example, the user who provided this scenario wanted to compute several types of averages, including a moving exponential average, of the closing prices retrieved from Yahoo! In his case, the scenario would incorporate Actions for the computations, which involved arithmetic, summation over sets, and even recursive functions. (Moreover, the user wanted to read data from a third site with Actions similar to those above, as well as read data from a spreadsheet.)

In order to describe these computations, the user provided several files that are publicly available at `www.getafreelancer.com/projects/Data-Processing-Data-Entry/Data-entry-copy-paste.html`

Retrieving data from tables involves indexing into the table based on date and, in a likely scenario variation, reformatting between `DD-Mon-YY` or `MM/YY` and whatever date format the spreadsheet uses for d_i.

Another variation could include stripping the commas out of the Volume returned by the web server before copying it into the spreadsheet.

Due to the large amount of data involved in this scenario, and the fact that much of it is historical, it would be inefficient to restart a macro from the beginning if it failed partway during execution. In that case, starting from the failed location might be very desirable, leading to a scenario variation.

Macro Maintenance

The Internet Archive's Wayback Machine contains copies of Yahoo! finance from several points back to 1998. Since then, the site has changed in every conceivable way, ranging from complete structural changes, to color and font changes, to widget name changes, and even to URL changes for the form's target. In the face of these waves of change, it seems likely that significant maintenance (and periodic rewrites) would be necessary. In particular, the user might need to add substantial error handling.

At a more basic level, it seems possible that once a user had recorded a macro for this scenario, then retrieving additional statistics would be desirable. For example, instead of retrieving just the Price-to-earnings and Price-to-sales ratios, the user might also want to retrieve the adjacent Price-to-book statistic.

Scenario Source

In January 2006, a user publicly posted a specification of this scenario in order to hire a freelancer to implement the scenario. In addition, this user has posted a number of similar jobs that also relate to financial analysis. The programmer who won the work claims to be skilled in "XML/XSLT, Python, Perl, and PHP".

Watcher for eBay — Typical User: Online shopper

Overview
Online auctions run for hours or days. During this time, the auction site is actively monitored by buyers and sellers (who are often also online shoppers, of course). Buyers are inclined to watch the items they are interested in; they often bid at the last minute to make sure that they win the auction. Sellers monitor auctions so they can manually enforce restrictions, such as banning bidders who have received a highly negative rating from sellers at previous auctions. Buyers and sellers access up-to-date auction information by visiting an eBay item page and refreshing the page (often several times per day).

One person has written a script that end users can configure to monitor an item auction. The script is a blend of three APIs: JavaScript, Google, and iGMonkey. To install it, users go to their personal Google portal (`http://www.google.com/ig`) and specify the URL where the script is located. Google portal instantiates the script to generate a new module widget and displays it. The user types an eBay item id into the widget, and then the widget automatically downloads the item's auction information. It caches this information and refreshes it at user-configurable intervals.

We have reverse-engineered this script's actions into a scenario below. Any macro tool that enables a user to create a functionally equivalent macro without any textual code would be an impressive tool, indeed.

Starting Conditions
User provides eBay item number.

Result
The item's image, name, current bid, auction end date/time, and high bidder have been retrieved.

Actions
1. Open the eBay web site.
2. Enter the eBay item number.
3. Retrieve the item's information.

Action Details

1 Open eBay web site, which is publicly accessible at `http://www.ebay.com/`

2 Enter the number to the search textbox and click the search button.

While a user might enter the item number through the form above, the script accesses the item page directly by appending the item number directly to the following URL:
`http://cgi.ebay.com/ws/eBayISAPI.dll?ViewItem&item=`

3 Retrieve item's image, name, current bid, auction end date/time. The script uses the following regular expressions:

- Current Bid: `bid:.*?<td.*?<b>(.*?).</b>.*?/td>`
- Item Title: `h1 class=.itemTitle.*?h1 class=.itemTitle.>(.*?)</h1>`
- End Time: `End.*?(<b>(.|\n)*?)</td>`
- Current Bidder: `bidder:.*?<a href=.*?>(.*?)<`
- Image Title: `<img title.*?JPG.*?>`

Variations

The script performs minimum error checking. If server doesn't return a valid item page, the script will not display anything. A more robust version of the script would provide an appropriate error message when an item is not found; there are various reasons why the script might fail to retrieve item information, such as an invalid item number or removal of the item's listing.

The scenario handles only one eBay item. A likely variation would involve watching multiple items. The result could be displayed in the Google portal, as above, or in a spreadsheet.

A macro for this scenario would differ from all the others in our corpus in several regards, though the details would depend on the scenario variation. First, the macro could trigger on page load every time that the user visits the Google portal. Second, it could avoid retrieving data from eBay every time that the macro runs; instead, it could cache values and performs periodic updates (every few minutes). Finally, variations would format the results as HTML for display inside of the Google portal as a visible module (below), or the HTML could be written to a file and used for another purpose.

Macro Maintenance

The Internet Archive's Wayback Machine contains copies of eBay from several points back to 1997. Since that time, eBay website has changed repeatedly in wide-ranging ways. For example, the eBay item search page now has a textbox and a category dropdown (Action 2) but only had a single textbox in 1998. Submitting the form will still bring user to the item description page.

The Internet Archive's Wayback Machine does not maintain copies of item detail pages, so it is not clear if the layout of the item page has changed. Due to the series of changes on the eBay website, periodic script maintenance might be necessary.

Scenario Source

The script is publicly accessible at `http://n79.org/modules/ebayWatcher.xml`

4. Requirements for Macro Tools

Our scenarios reveal various requirements for effectively enabling users to record and replay web macros. We have grouped these requirements into the same categories outlined in Section 1.

Macro tools have already begun to tackle some of these requirements. For example, Robofox permits conditional execution of operations, automatic triggering of scripts (so it automatically runs at certain times of the day or when the user visits certain URLs), and assertions that can perform "sanity checks" to detect when a macro might be failing [8]. Many other requirements remain.

4.1 Triggering macros

All scenarios involve some starting conditions, so the corresponding macros should not begin to execute until those starting conditions are met. The characteristics of these conditions vary as follow.

4.1.1 On-demand execution

Most scenarios begin after a conscious action by the user. These include scenarios that are driven by spreadsheets (e.g.: Currency Conversion) and those that perform lookup operations to help the user fill out a web form (e.g.: Per Diem Lookup). The scenario can begin when the input data are available.

When a macro uses a spreadsheet as input, and then writes results back to the spreadsheet, it would be helpful if the macro player provided buttons inside of Excel so that the user could open up the spreadsheet and play the macro. Another option would be to extend the spreadsheet formula language so that users can enter formulas that retrieve information from the internet; this is the approach taken in A1 [6]. The new values would appear in the spreadsheet right before the user's eyes, thereby facilitating testing and debugging. For similar reasons, corresponding buttons would be appropriate in the browser for when a macro uses form fields as input, and then writes results back to the web form.

4.1.2 Scheduled execution

In scraping scenarios, the input data come from a web site, so the arrival of fresh data could occur at any time. Consequently, these scenarios might benefit from scheduled operation of macros, in a sort of "polling" process. In fact, the Person Locator Scraper specifically contains actions that help ensure that the scraper could run repeatedly with no user intervention.

The macro tool might provide a user interface so that users could schedule playbacks. Alternatively, it could offer a command-line version so that users could schedule playbacks using the facilities provided by the operating system.

4.1.3 Event-based triggers

The Watcher for eBay demonstrates an interesting opportunity for triggering macros. If a macro's output can be formatted as HTML, then the macro tool might allow the user to register the macro on a page load event for a certain page. The macro player could then execute the macro and append the output (as HTML) to the page's HTML structure.

In fact, this would allow the macro to inject HTML for buttons, links, and other widgets into the page. Registering new event handlers on these widgets and linking additional macros to the event handlers would effectively create an application user interface. This is the approach taken by GreaseMonkey [13], though that platform requires users to write JavaScript rather than demonstrating actions via PBE.

4.1.4 Subroutines

None of our scenarios use other scenarios as subroutines, but it is not difficult to imagine situations where one macro might use another macro as a subroutine. For example, macros for variations of the Path to Procurement and Stock Analysis scenarios might use a variation of a Currency Conversion macro in order to localize prices. In addition, if an organization had multiple staff directories, then a macro might call several Peoplesoft Scraper or Staff Lookup macros and then merge the results.

4.2 Using objects on web pages

The dynamism of the web makes for a very different execution environment than in traditional software development. When a traditional program executes within a computer, a basic fetch-execute CPU can blindly plod through a list of operations. The register hardware never moves around; the memory hardware never reconfigures without a power cycle.

In contrast, robust macro players must be much more adaptive: find the currency amount (what color is it today?), find the "source" textbox (where is it today?), reformat the currency and copy to the textbox, click the submit button (no submit button today?—perhaps an "enter" keystroke will do?), wait for the site to load (it is slow today), try to find the result within the page, and so forth.

Consequently, it appears that achieving a reasonable level of robustness will require a fairly abstract representation for macros so the macro player can adaptively apply the user's intention at runtime even if it detects that the page has undergone significant changes since the macro was recorded. At the very least, the macro player should detect changes and notify the user, rather than doing something wrong.

4.2.1 Text snippets

As demonstrated by all scenarios, a basic requirement of web macro tools is that they should be capable of retrieving web pages from servers, and then extracting portions of the pages' text. The text is sometimes wrapped in an HTML tag of its own (as with the bright red currency quantities in the Currency Conversion scenario). However, the text may also be buried within a larger section of text with absolutely no HTML markup to delimit the target text (as with the currency quantities in Action 7 of the Per Diem Lookup).

4.2.2 Tabular information

Several scenarios involve interpreting tabular information and retrieving data from one or more rows or columns. For example, in Per Diem Lookup and Stock Analysis, the macro must retrieve data from specific rows that have an appropriate date in the leftmost cell. Achieving this requires identifying the table within the HTML, parsing it into keyed records, filtering records based on whether their respective keys match certain criteria, and then retrieving fields within those records for use in computations.

Web browser plug-ins have recently become more sophisticated in their handling of tabular data. For example, Sifter reads HTML and finds repeating structures (which the tool represents with XPATH expressions). For each cell or section within that structure, it uses hardcoded parsers for common data types. This enables the tool to support sophisticated queries within the user interface [5].

The Sifter tool does not provide a macro facility for automating these queries, and it only uses a limited set of data types (e.g.: numbers, text, and dates). It is unclear whether Sifter could handle the composite keys in the Per Diem Lookup. For example, Action 3 looks up per diem rate based on city and date range, Action 6.1 looks up county name based on city and state, and Action 6.2 looks up per diem rate based on date range and county. Each of these essentially indexes the table using a composite key, where part of the key is sometimes missing.

4.2.3 Other HTML structures

The Watcher for eBay scenario demonstrates injection of HTML into a web page. The ideal macro tool will allow users to reformat macro output into a textual or HTML format, possibly using a template that the player fills in at runtime, and then inject it into a web page.

4.2.4 Adaptation to changing page layout

Site changes could cause two types of problems when picking data out of HTML.

- If macro players only find strings of text based on one or two visual characteristics of the text, then changes in the font, color, and other visual attributes could break a macro. For example, if a Currency Converter macro tries to find the second red text on the page (which is the output value in US dollars), and the site evolves so this text becomes orange, then the macro will be unable to find the value.

- If macro players only find strings of text based on structural characteristics of the text, then evolution in page layout could break a macro. For example, if the Package Tracker macro tries to find the result table based on nesting of HTML tags ("XPATH" targeting), and the site evolves so that the results are moved inside of another table, then the macro will be unable to find the values. (For example, the Western Union site once swapped the location of "Cancel" and "Ok" buttons. They did not have unique IDs. Any macro player that depended purely on XPATH expressions would have failed to adapt to this change.)

The Internet Archive's Wayback Machine and comments in sites' HTML code reveal that many sites involved in our scenarios have evolved over the years. Despite this, this we suspect that this evolution would not have prevented human users from finding data on the pages. Even though humans cannot see the nesting of tags and cannot use XPATH expressions to find data, they still can adapt. The goal is to find ways of helping macros adapt to page evolution as well as humans do.

In order to adapt to changing page appearance, existing macro tools such as Chickenfoot have incorporated a number of techniques, such as keyword matching and text constraint matching [1] as well as English-like statements by the user that help to guide the tool to the right location on the page [11]. However, they have yet to incorporate all of the cues that humans can use to locate data within the page:

- Humans can remember where the data appeared relative to the page during the last visit, so they can search the same location (i.e.: X Y position on the page) during later visits. If the data is not in the expected location, the human can fall back to searching the page in its entirety.
- Humans can remember the size and color of data that they saw during the last visit, and they can quickly locate text with similar visual characteristics, such as font and color [24]. If they cannot find the text with these criteria, then they can thoroughly search the entire page.
- Humans often know the usual format of data that they want. As they search the text, they can ignore words that do not have the expected format. For example, numbers should have digits, and ticker symbols should have capital letters.
- Humans often know the semantics of words surrounding the text. They can ignore data values if the surrounding text indicates that those values do not correspond to the desired data.
- Humans can rely on various conventions for interpreting pages. For example, tables' top row and left column often contain labels rather than actual data; likewise, the top and left sides of pages usually contain navigation rather than actual content. Another convention is that when lists span multiple pages, there is usually a link containing the word "Next" to reach the next page.

As noted above, macros that only rely on structural cues are likely to break if the site evolves. Macro tools are only beginning to take advantage of semantic cues [3]. Most do not yet take advantage of sophisticated new machine learning techniques that would help the tool to learn how to find data in the new HTML [9].

4.2.5 Web form widgets

Most scenarios involve getting or setting the values of various web form widgets, including textboxes, dropdowns, and radio buttons. These get/set operations resemble DHTML in that they involve manipulating objects within the page's structure. However, they may be somewhat simpler to automate than arbitrary DHTML operations because widget get/set operations never modify the page's structure.

In many cases, the macro tool could compose http operations directly, which would reduce the need for manipulating widgets. However, the macro player will still need to support widget get/set operations since scenarios like Per Diem Lookup require reading inputs and writing outputs from/to a form that the user has opened in another browser window.

4.2.6 Adaptation to changing form fields

Many macro players are browser proxies that perform http operations by contacting the server directly (rather than contacting the server indirectly by rendering pages, filling widget values, and clicking a submit button). The direct http approach requires the macro player to open a socket and transmit a list of variable

names and corresponding values. These variable names must match the names that the server is expecting; in particular, the names must match the names of widgets on the web form that the player is emulating.

Therefore, evolution in the names of form widgets can break macro players that use the direct http approach. This has occurred in a couple of scenarios. For example, in July 2005, a programmer added a new hidden field inside the Path to Procurement web site; presumably the server software was also modified so that it now uses this new variable. Any macro recorded prior to the addition of this field would not contain any instructions for transmitting a variable with that name. Consequently, if the new version of the server software requires the presence of this hidden field, then the server might not perform as anticipated during macro playback.

Evolution in the valid *values* of form widgets can also break macro players that use the direct http approach. For example, in the Currency Converter, the code for a Bulgarian Lev has changed from "BGL" to "BGN" since 1998. If a user recorded a macro prior to this change, and the player uses the direct http approach, then the player would still keep sending the old value, which the server might not understand. In other words, the player would fail to adapt the user's intent to the site's new structure.

One way to prevent these errors would be for the macro player to check at runtime whether any new fields have been added or modified in the form. If fields with no visual representation have been added, then the macro player could silently append the new fields' default values to the http operation (since this is what would happen if a human user performing the scenario encountered the same situation). If fields with a visual representation have been added or modified, then the macro could alert the user and let the user decide whether to modify the macro accordingly.

Another way to prevent failures would be for the macro player to take a more indirect approach to contacting the server: render the page and manipulate widgets like a human would. Unfortunately, this approach has its own problems.

First, manipulating widgets requires finding them, which can be difficult if the widget names have changed. Some macro tools are more immune than others to this problem. For example, Chickenfoot locates widgets based on text near to widgets [1], and this text might not change even if widget names change.

Second, evolution in the valid values of form widgets can also break macro players that use this indirect approach. It may be possible for an indirect http macro player to adapt to this. For example, the macro player could look through the list of currencies in the Currency Converter scenario and locate the new code for the Bulgarian Lev. Achieving this sort of adaptation may require taking advantage of semantic and formatting cues of the sort described in the previous section.

4.2.7 Adaptation to changing URLs

Most scenarios start with "go to the following URL," but like page structure, page locations evolve. For example, the government's Per Diem Lookup used to be located on the www.policyworks.gov server and has since moved to the www.gsa.gov page discussed in the scenario. (The page structure also changed slightly.) Any macro programmed to use the old URL might struggle to locate the new page.

Fortunately, the webmaster of the old server put up a web page telling users that the old content has moved and providing a link to the new location.

While some sites post pages like this when content moves, other sites use an HTML META refresh tag to redirect the browser to the new location. Some sites use http redirects.

In any case, the macro player could detect that the page has changed significantly. For example, Robofox automatically inserts assertions after every operation to determine whether the page's structure matches the tool's expectations [8].

However, there is no reason to stop there. If a macro player detects that the page has changed considerably, it could examine the page to find a new URL. With the user's permission, it could then retrieve the content at that new URL and see if the structure matches the expected structure at the old URL. If so, the player may be able to update the macro and continue.

4.3 Reading/writing data outside pages

All scenarios involve reading and writing data from the browser, but some also involve reading and writing from other locations such as spreadsheets.

Even though our scenarios did not uncover them, we are aware that there are a number of other systems where web-related data often are located. These include databases, word processors, RSS feeds, web services, and email servers. The macro tool's architecture should probably allow future expansion to accommodate additional data locations like these, since some users may want to transport data between these systems and the web browser.

4.3.1 Browser APIs

It may be desirable to display output within the browser, but outside the web page.

For example, the Currency Conversion scenario mentioned a possible variation in which the user wants to highlight a number within a web page's text and feed that number into the currency converter. The macro would infer the correct source currency from the web page's URL, convert it to US dollars, and then display a popup window with the amount in dollars.

To support this scenario variation, the macro tool must be able to read the currently highlighted text and the current URL, then display results in a popup.

4.3.2 Spreadsheets and other files

Several scenarios involve reading data items from a spreadsheet, using each data item to perform lookups on the web, and then writing the results back to the spreadsheet. In general, each data item is in its own spreadsheet row, and macros would have the form, "For each row R, read the data item d_R, look up n values $\{f_{1R}, f_{2R}, \ldots, f_{nR}\}$, and write these values alongside d_R in row R."

In addition, the Person Locator Scraper writes an XML document, which only the most recent web macro tools support (and, even then, they require users to write at least some code in a sophisticated language such as SmallTalk [12]). To support this scenario, the macro recorder might allow the user to define a template that the macro player would instantiate and fill in at runtime.

4.3.3 Parameters containing user input

Although most macro input comes from the sources described above—web pages, spreadsheets, and so forth—the user may want to parameterize the macro and manually provide parameter values when executing the script.

For example, several scenarios involve authentication. When the user demonstrates the example and types a username and password, should the macro record the username and password, essentially hard-coding these as part of the macro, which could inhibit sharing the macro with other users? Or should the macro represent the username and password as parameters that are undetermined until runtime, which could be a hassle for the user when executing the macro?

Our scenarios suggest a reasonable default behavior when the macro tool is faced with questions like these.

In every scenario where submitting a form causes the server to set hidden state (as during authentication, or when selecting a country in the Package Tracker), the correct behavior is probably for the macro recorder to ask the user if parameterizing the values is appropriate. That way, the user can control whether the macro contains sensitive information such as a password.

However, in most other situations, our scenarios suggest that the macro tool could simply record the user's input by default. These sorts of input take the form of typing URLs, clicking the mouse, typing a search query, and so forth. If the macro supports inspection and maintenance of macros (as discussed by the last category of requirements, below), then the user can always view the inputs that have been recorded into the macro and, if desired, can turn those inputs into parameters by deleting the hard-coded value.

It may be beneficial to have a type of "sticky" input parameter. For these parameters, when a value is set for one execution of the macro, the macro tool would that value and use it as a default value during the next execution of the macro. The user could always override the default for a given execution, thereby changing the default value for subsequent executions. This sort of "sticky" behavior would be beneficial for particularly complex inputs. For example, a variation of the Peoplesoft Scraper would accept a set of regular expressions that control which records the macro retrieves. We speculate that the system administrators using such a tool would appreciate not having to retype these regular expressions unless there was a need to change the macro's behavior.

4.4 Transforming data

Our scenarios demonstrate that using data from the web involves much more than simply unescaping HTML characters (e.g.: from "&" to "&"), but rather involves a number of more sophisticated transformations.

4.4.1 Reformat to equivalent value

The Per Diem Lookup scenario involves a significant amount of reformatting. For example, matching up choices in the image map with values in the expense report involves reformatting between state names and state abbreviations. In addition, the scenario involves reformatting dates from `MM/DD/YYYY` to `Month D`. Finally, it involves simple string manipulations, as when stripping the literal "County" substring from the value returned by Action 6.1 and capitalizing the county name for comparison to other county names in Action 6.2.

Other scenarios also involve small reformatting operations based on the semantics of the data. Examples:

- The Stock Analysis reformats dates from `MM/DD/YYYY` to `DD-Mon-YY`.
- The Staff Lookup repairs phone numbers from `###-### ####` to `###-###-####` format and strips spurious spaces from email addresses.
- The Person Locator Scraper transcribes characters with accents and umlauts to nearly-equivalent lower-ASCII characters.
- The Person Locator Scraper interprets status information for each person record in order to set a Boolean flag indicating if the person has been found. For example, if the person is known to be "Hospitalized" or "Deceased", then it sets the Boolean to true. This essentially involves running the value through a small lookup table.

Operations like these transform a data value to another value that is semantically "equivalent" for the purposes of the scenario. Macro tools could provide a way for users to specify transformations like these. In addition, the tools could intelligently perform commonly occurring transformations, such as those involving dates.

4.4.2 Extraction of values' parts

Other operations extract part of a value and discard the rest. For example, a Per Diem Lookup macro would extract the year from a `MM/DD/YYYY` value. In a later action, it would extract the month and day. It also would also extract the city and state from a `City, ST` value.

4.4.3 Combination of values

A few scenarios involve combining values. The mode of combination depends of the types of the values.

- The Stock Analysis scenario mainly involves numbers, which the user naturally wants to combine using arithmetic and algebra. Likewise, the Per Diem Lookup adds two numbers to generate a default per diem rate.
- In the Per Diem Lookup, dates are compared to one another in order to determine the appropriate entry in a table of per diem rates.
- In the Watcher for eBay, scraped values are treated as strings and concatenated with HTML.

4.5 Executing control structures

Macro recorders must be able to execute three types of operations: primitive, looping, and conditional. As discussed in the previous categories of requirements, the primitive operations include operations required for manipulating the web browser (such as clicking links). In this section, we consider looping and conditional operations.

4.5.1 Looping operations

Sometimes the target of an operation is a set of objects. For example, the Per Diem Lookup ends with an action that picks two numbers out of the text and adds them together. When the user demonstrates this addition operation, should the macro recorder infer that the user wants to add together just these two numbers? Or should it infer that the user wants to add together any number-like quantities that appear in the text? This second choice would allow the macro to gracefully adapt if the government per diem rate contains a third component some day.

Scenarios demonstrate other repetitions of an operation on each object in a set. For example, several scenarios repeat actions for each row in a source spreadsheet. In addition, the scraper scenarios repeat read operations for each page in a list of records. Finally, many scenarios perform a read operation on each row in a table. Loops of these types are sometimes represented in textual languages with a "foreach" construct.

The Watcher for eBay performs certain operations periodically. In a textual language, a loop of this type might be represented as "while(true) {run script; sleep;}" Support for a general "while(condition)" construct might also be useful for polling web sites until a condition is met (such as polling the Hurricane Katrina web site in the Person Locator Scraper to see whether new whether new records have appeared) , and for representing loops over sets of objects as discussed above.

4.5.2 Conditional operations

Sometimes the scenario involves certain actions depending on the conditions at runtime. For example, Staff Lookup involves different actions depending on whether the server returns zero results, one result, or multiple results. How will the user specify these three possibilities to the macro recorder, when a single demonstration can only exemplify one of these three conditions? Presumably the web macro recorder will need to incorporate multiple examples, just as non-web macro recorders such as Eager have done [2].

Other scenarios exemplify conditionals, as well:

- Path to Procurement – if the item is the last item, then click one link; otherwise, click the other
- Per Diem Lookup – if the per diem look up based on state & date succeeds, then the scenario terminates; otherwise, it proceeds to Action 4. If the county look up based on city & state fails, or if the per diem look up based on county & date fails, then the scenario results in using a default value.
- Person Locator Scraper – if the timestamp file is present, then look for where new data starts
- Stock Analysis – if the stock's data is unavailable for the specified year on MSN, then use the stock's data for the current year as a default

4.6 Recovering from failure

In some cases, the macro tool will be unable to prevent failure. For example, the computer might lose its network connection, or the server might crash, or the page might have evolved so much that the macro tool cannot automatically figure out how to use the new page. In these cases, the macro player must help the user recover from the failure as gracefully as possible.

4.6.1 Partial restarts

If a macro fails halfway through a scenario, it may be safe to restart the macro from the beginning. This is typical with scraping and lookup scenarios. For example, if the Staff Lookup successfully retrieves data for 50 of 100 co-workers, but then the server crashes, then there is no harm in restarting the macro later.

Of course, repeating work is wasteful. For example, the site targeted by the Person Locator Scraper has over 1000 pages. If a macro delayed 5 seconds between http requests (to avoid overwhelming the server), then it would require over an hour to run. Reprocessing a significant fraction of the data would be unnecessarily tedious, which may explain why the scraper's author chose to insert the scraper's timestamp-checks.

Moreover, some operations are not safe to repeat, due to side-effects. For example, the Scraper for CMS scenario inserts records into the target site. Repeating these operations would probably result in duplicates.

Consequently, the macro tool should track how far macros proceed. That way, if a macro fails, then the user has the option of doing a partial restart—that is, restarting the macro from where it left off.

How might partial restarts be implemented?

If an operation can be repeated safely, then web developers call that operation "idempotent" (in analogy to the mathematical concept of an operator that does not modify the operand). Idempotent interactions with the server are generally performed with GET requests; non-idempotent interactions are generally performed with POST requests.[3]

Therefore, one approach to achieving partial restarts would be for the macro player to log operations (and cache the http results) as they occur. If a failure occurs, then the player could break the log into transaction-like segments, with POST operations identifying the boundaries between segments. All POST operations prior the failure should not be repeated, and any necessary data from GET operations prior to those POST operations should be served from the cache. Any GET operations after the last POST operation could be re-executed if desired. In effect, the partial restart would begin from the moment after the last POST operation.

Of course, this approach would not serve perfectly in every circumstance. For example, authentication steps that sent a permanent or session cookie would have to be performed during the restart, even though they are POST operations. There are probably other occasions when the macro tool should let the user override the default approach described above. To help the user make an informed decision about how to proceed, the macro tool would still need to log operations and cache the http results.

4.6.2 Exception handlers

The macro tool should also allow the user to specify how to handle exceptions as they occur. In addition, the macro tool should help users add new exception handlers as the user adds new examples, as these examples will uncover new response patterns by the server. As described above, several scenarios involve conditionals that cope with differences in how the servers respond to different inputs.

For example, tools should enable users to create an assertion that fires at runtime if data looks out of the ordinary or if the web page's structure seems to have changed in a way that the tool cannot automatically

---

[3] http specification, `http://www.w3.org/Protocols/rfc2616/rfc2616.html`

handle. An exception handler would alert the user to the problem and ask for guidance. Even a simple exception handler of this type might help users to trust that their macros will perform properly.

Moreover, if users could attach assertions and exception handlers to existing macros, then they could reuse another person's macro and add assertions to help ensure that the macro would behave as desired. In short, assertions and exception handlers may be essential for improving the reliability and sharability of macros.

4.7 Supporting macro maintenance

Records in the Internet Archive show that many of the sites involved in our scenarios have evolved significantly over the years. In some cases, this evolution might have broken macros automating the scenarios. Therefore, macro tools should support the maintenance of macros by end user programmers.

4.7.1 User-understandable representation

Before a user can perform maintenance activities, it is first necessary to understand the macro's internal structure. For example, Robofox displays a list of statements, each representing a step taken by the macro [8].

We believe that such a representation will probably prove extremely valuable for other activities, in addition to maintenance. For example, if one user offers to share a macro with another user, the recipient may want to examine the internal structure of the macro before executing it, in order to determine whether to trust the macro. With a user-understandable representation, it would even be possible for users to post their macros in a public repository so that other users could browse through the repository, view the macros' internal structure, and then select an appropriate macro for a task at hand.

4.7.2 Editable macros

Another basic requirement for maintenance is the ability to make changes to existing macros. Desirable edit operations include deleting operations, adding operations, changing operations, wrapping operations in loops, and all of the other types of edits that are currently supported in textual editing environments. When designing a macro language and editor for end users, it is important to consider a number of issues that can inhibit usability, such as the difficulty of making small changes to programs in visual languages (viscosity) [16].

4.7.3 Features for debugging

Many professional programmers have come to rely on various sophisticated debugging services within the development environment. These services include traces, breakpoints, assertions, step-by-step execution, and runtime variable inspection. Macro tools have only recently begun to provide similar features; for example, Robofox highlights objects in red as a program executes, and it supports assertions [8].

When carrying over debugging ides from professional programming tools to end user programming tools, it is important to ask whether it is possible to provide enhancements that will make debugging easier or more effective. For example, Ko implemented the WhyLine for the Alice end user programming environment; this WhyLine can answer questions about why program behaviors did or did not occur during execution. This tool reduced the time for debugging activities by approximately 90% [7], and something like it might be highly helpful to users of web macro tools.

4.7.4 Maintenance at runtime

A macro might break because site evolution prevents the macro player from picking data out of the HTML, getting or setting widget values, or following URLs. However, the changes leading to the broken macro might have been minor:

- Perhaps the data changed from red to orange.
- Perhaps the widget was renamed.
- Perhaps the page moved to a different URL.

In such cases, it would be desirable if the macro tool provided a way for the user to modify the macro to fix it. For example, the user could highlight the data or widget so the tool could relearn how to find the data or widget.

When site changes are more significant, the tool may need to provide mechanisms to add new operations to existing macros. For example, the government site in the Per Diem Lookup sometimes displays new regulations on how to use the site. The macro tool could provide a way for the user to specify that the macro should check at runtime whether these regulations have changed—and, if so, to enter a maintenance mode so the user could incorporate the regulations into the macro.

5. Opportunity for Growth

Web macro tools have evolved significantly over the past decade, but they have not yet reached their full potential. The requirements that we have uncovered in our scenarios provide a benchmark for measuring future improvements in macro tools.

In a similar way, the evolution of our scenario corpus is not complete, either. We are setting up a wiki[4] where we and other researchers can read scenarios and contribute new scenarios. Over time, we hope that this corpus will continue to grow as macro tools meet existing requirements and we as a community discover new requirements.

6. Acknowledgements

We are grateful to Mary Shaw and other EUSES members for helpful discussions. This work was funded in part by the National Science Foundation under ITR grant CCF-0325273 (via the EUSES Consortium) and by the National Science Foundation under ITR grants CCF-0438929 and CCF-0324861. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

7. References

[1] M. Bolin, M. Webber, P. Rha, and T. Wilson. Automation and Customization of Rendered Web Pages. *UIST '05: Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology*, 2005, pp. 163-172.

[2] A. Cypher. EAGER: Programming Repetitive Tasks by Example. *CHI '91: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1991, pp. 33-39.

[3] A. Faaborg and H. Lieberman. A Goal-Oriented Web Browser. *CHI'06: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2006, pp. 751-760.

[4] J. Fujima, A. Lunzer, and Hornbae. Clip, Connect, Clone: Combining Application Elements to Build Custom Interfaces for Information Access. *UIST'04: Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology*, 2004, pp. 175-184.

[5] D. Huynh, R. Miller, and D. Karger. Enabling Web Browsers to Augment Web Sites' Filtering and Sorting Functionalities. *UIST'06: Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology*, 2006, pp. 125-134.

[6] E. Kandogan, E. Haber, R. Barrett, A. Cypher, P. Maglio, and H. Zhao. A1: End-User Programming for Web-Based System Administration. *UIST'05: Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology*, 2005, pp. 211-220.

[7] A. Ko and B. Myers. *Designing the Whyline: A Debugging Interface for Asking Questions about Program Behavior*. *CHI '04: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM Press, 2004, pp. 151-158.

[8] A. Koesnandar, S. Elbaum, and G. Rothermel. Building Dependable Web Macros with Robofox. Technical Report TR-UNL-CSE-2006-0010, Department of Computer Science and Engineering, University of Nebraska -- Lincoln, August 2006.

[4] `http://softwaresurvey.cs.cmu.edu/wmcorpus.html`
`Or contact authors for a new location.`

[9] K. Lerman, S. Minton, and C. Knoblock. Wrapper Maintenance: A Machine Learning Approach. *Journal of Artificial Intelligence Research*, Vol. 18, 2003, pp. 149-181.
[10] H. Lieberman (Ed.). *Your Wish is My Command: Giving Users the Power to Instruct their Software*. San Francisco: Morgan Kaufmann, 2000.
[11] G. Little and R. Miller. Translating Keyword Commands into Executable Code. *UIST'06: Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology*, 2006, pp. 135-144.
[12] A. Lunzer and K. Hornboek. RecipeSheet: Creating, Combining and Controlling Information Processors. *UIST'06: Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology*, 2006, pp. 145-153.
[13] N. McFarlane. Fixing Web Sites with Greasemonkey. *Linux Journal*, Vol. 2005, No. 138, 2005.
[14] R. Miller and B. Myers. *Creating Dynamic World Wide Web Pages by Demonstration*. Technical Report CMU-CS-97-131/CMU-HCII- 97-101, Human Computer Interaction Institute, Carnegie Mellon University, Pittsburgh, PA, May 1997.
[15] *A Nation Online: Entering the Broadband Age*, US Department of Commerce, Economics and Statistics Administration, National Telecommunications and Information Administration, 2004.
[16] J. Pane and B. Myers. *Usability Issues in the Design of Novice Programming Systems*. Technical Report CMU-CS-96-132, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, August 1996.
[17] R. Potter and D. Maulsby. A Test Suite for Programming by Demonstration. In *Watch What I Do: Programming By Demonstration*, 1993, pp. 539-592.
[18] A. Safonov, J. Konstan, and J. Carlis. Towards Web Macros: A Model and a Prototype System for Automating Common Tasks on the Web. *CHI'99: Proceedings of the 5th Conference on Human Factors & the Web*, 1999.
[19] C. Scaffidi, A. Ko, B. Myers, M. Shaw. Dimensions Characterizing Programming Feature Usage by Information Workers. *VL/HCC'06: 2006 IEEE Symposium on Visual Languages and Human-Centric Computing*, 2006, pp. 59-62.
[20] C. Scaffidi, B. Myers, and M. Shaw. Trial by Water: Creating Hurricane Katrina "Person Locator" Web Sites. *Leadership at a Distance* (S. Weisband, ed), Lawrence Erlbaum, 2006, to appear.
[21] C. Scaffidi, M. Shaw, and B. Myers. Estimating the Numbers of End Users and End User Programmers. *VL/HCC'05: 2005 IEEE Symposium on Visual Languages and Human-Centric Computing*, 2005, pp. 207-214.
[22] C. Scaffidi, M. Shaw, B. Myers. Games Programs Play: Obstacles to Data Reuse, *2nd Workshop on End User Software Engineering (WEUSE)*, 2006.
[23] A. Sugiura and Y. Koseki. Internet Scrapbook: Automating Web Browsing Tasks by Demonstration. *UIST'98: Proceedings of the 11th Annual ACM Symposium on User Interface Software and Technology*, 1998, pp. 9-18.
[24] A. Treisman and G. Gelade. A Feature-Integration Theory of Attention. *Cognitive Psychology*, Vol. 12, No. 1, 1980, pp. 97-136.