

Technical Report

Kent State University/Lemur Remote Toolkit (KLRT)¹:

A toolkit for remote collection of client-side search logs

February 20, 2012

Catherine L. Smith*
Guan Wang^

Kent State University
* School of Library and Information Science
^ Department of Computer Science

¹ The toolkit is an adaptation of the Lemur Query Log Toolbar, developed at The Center for Intelligent Information Retrieval (CIIR) at the University of Massachusetts, and the Language Technologies Institute at Carnegie Mellon University, and available on SourceForge at <http://sourceforge.net/projects/lemur/files/Lemur%20Query%20Log/>

1. INTRODUCTION AND OVERVIEW

This report describes the Kent State University/Lemur Remote Toolkit (KLRT). Our objective with this report is to document the components and functionality of the toolkit, and to provide general information on the ways in which it can be adapted for various research designs. The target audience for the report is researchers who are familiar with the use of client-side logging for the study of interactive search behavior. The report will be most useful for those interested in deploying the toolkit, including those who will be adapting and implementing the code.

The KLRT is a modification and extension of the Lemur Query Log Toolbar 2.1 (LQLT)². The toolkit facilitates collection of client-side data during search interaction in the Firefox 3.6 browser (FF). The toolkit includes (1) a browser add-on that logs browser events, (2) a back-end that collects and manages uploaded log data, and (3) an optional website structure that may be coupled with the add-on in order to control the add-on and administer study protocols.

We developed the KLRT to meet several objectives. Most importantly, we wanted a reusable tool for data-collection that we could adapt for different types of research designs. For designs that use assigned search tasks, we needed a way to couple administration of the study with the functionality of the LQLT add-on. We did this by designing a website structure for control of add-on functions, and adding new functions. We also needed tools we could deploy in completely remote studies (Smith, 2011), so that research subjects could easily download and install the add-on without assistance. For remote studies, it was important to have one installation procedure that worked easily across all Windows operating systems, so we built the toolkit using the Firefox version of LQLT 2.1³. Finally, we needed to provide subjects with reliable controls for protecting their privacy throughout a study. We did this by adding new functionality and interface components that enable control of the add-on.

The KLRT consists of three main components: a browser add-on, a website, and a backend:

Add-on. The add-on comprises (1) client-side logging functions, (2) a toolbar interface that allows subjects to control logging and upload functions, and (3) control functions that work with corresponding website controls. The add-on can be adapted for different study designs, however the current version is not truly modular, thus it is not configurable in the traditional sense. The system may be adapted by enabling or disabling optional functions within the code. We've reused functions available in LQLT 2.1, however, we have made modifications and added several new functions. The current 4.0 version of the add-on does not run on newer releases of FF (4.0 and above; see section 4.5 for discussion). We discuss the add-on in section 3.1.

Website. The toolkit includes a website structure for running remote and laboratory studies. The KLRT add-on includes a set of *website control functions* that may be used to administer a research protocol. For example, these include: (1) subject identification, (2) validation of add-on installation, (3) controlled add-on activation, (4) collection of task-related data via forms, (5) automatic initiation of log uploads, and (6) validation of add-on uninstall. The website structure has four basic parts: (1) add-on installation, (2) instructions for study protocols, (3) an interface for administration of study protocols, including assignment of experimental treatments and search tasks, and (4) instructions for uninstalling the add-on (see Figure 1, below). The website and control functions may be used in various combinations, and may be customized to meet

² <http://www.lemurproject.org/querylogtoolbar/>

³ A version of LQLT 2.1 is available for Internet Explorer

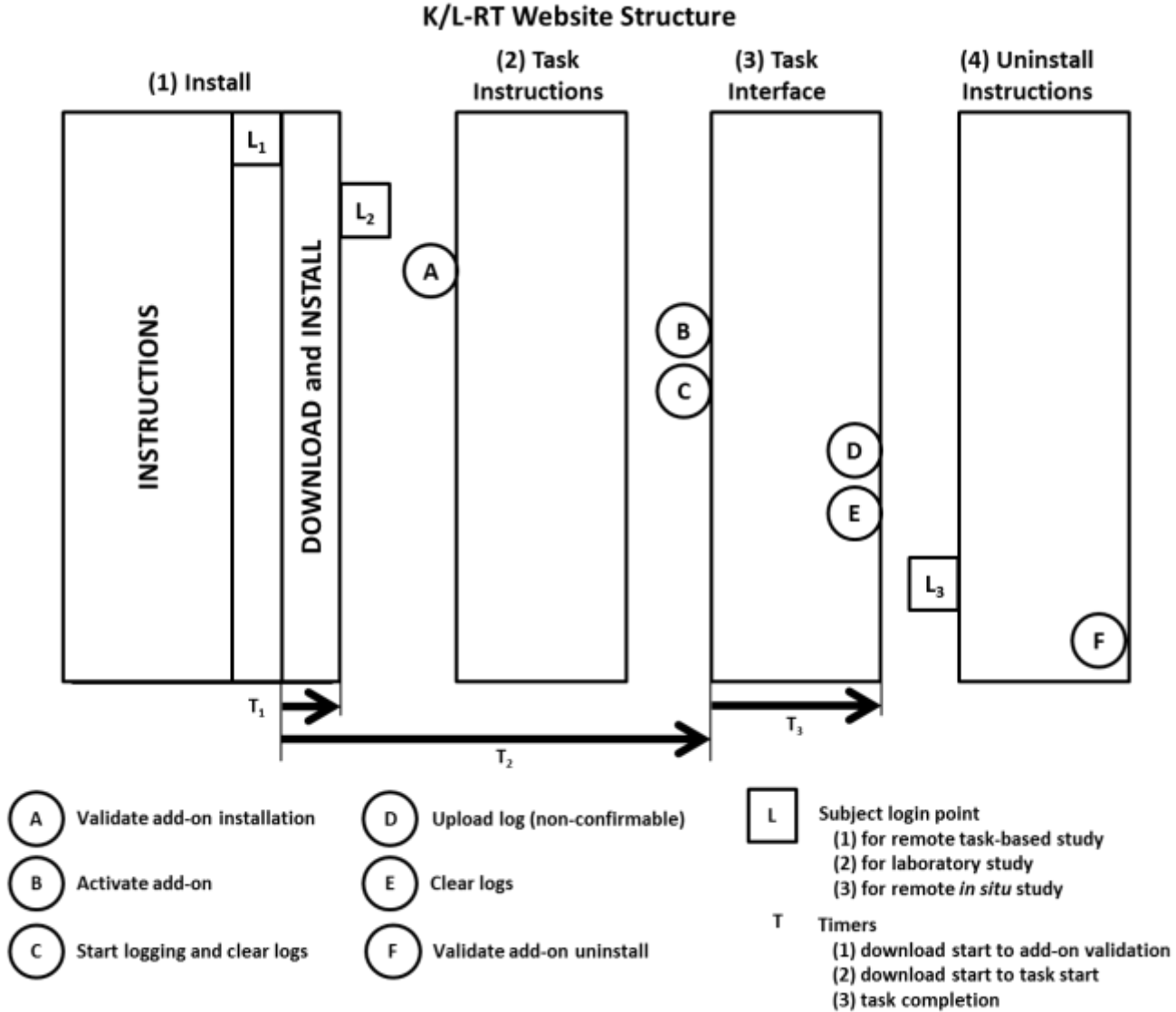


Figure 1. KLRT Website Structure

specific protocol requirements. The website uses php and a MySQL database. We discuss the website in section 3.2.

Backend. The backend is a server application running a MySQL database and optional parsers. The KLRT uses the backend functions provided with the LQLT, however, we have modified the backend. As currently configured, the backend works well for small-to-moderately sized studies, but it is not intended for web-scale data collection. The backend is discussed in section 3.3.

The remainder of the report has four parts. Section 2 scans related prior work. In section 3, we describe the components of the KLRT in detail. Section 4 concludes with a brief discussion of limitations of the current version of the toolkit, and directions for future development. References are listed in section 5.

2. BACKGROUND and PRIOR WORK

For many research questions in human-computer interaction/information retrieval (HCIR), access to client-side logs is essential. Client-side logs are contrasted with *server-side logs*, which only record requests made to a central server. Client-side logs record interaction on the users' machine, as well as requests made to *any* server. Client-side logs may be the sole source of data in a study. However, when supplemental data such as eye-tracking are required, logging functionality becomes a component of a larger more comprehensive research system, such as the Poodle framework of Bierig, Gwizdka, and Cole (2009).

In this section of the report, we briefly scan prior work describing various types of client-side logging systems, and their deployment in studies with various research designs (see Appendix 1, Table 1). Our objective is to place KLRT in the context of these systems and studies. First, we discuss several characteristics of the systems. We then review the ways in which the systems have been deployed, and the basic features of the research designs in which they have been used.

2.1. System characteristics.

Client-side research systems may be categorized on the basis of the types of data collected, the method of implementation in the browser, and for experimental protocols, on the type of administrative control mechanism used (see Appendix 1, Table 2).

2.1.a. Types of data collected.

A client-side log may contain only clickstream data (requested urls, with timestamps and identifiers), or it may also contain more detailed interaction data. Typically, the log contains records of browser *events* (e.g. page loads, scrolling, button presses), however, data about the state of the machine and other applications may also be captured (e.g., screenshots of a machine's display). In many studies, logs are supplemented with additional data, which may be obtained directly from research subjects (e.g., via questionnaires), or it may be collected from the environment (e.g., html of requested webpages).

2.1.b. System implementations.

Browser add-ons. Many studies use data collected from commercial *toolbar* add-ons. Typically, commercial toolbars are distributed by search engines, either directly, or in partnership arrangements with application providers (Baker, 2004). Commercial toolbars provide enormous volumes of clickstream data, which is generally collected without supplemental information. Beyond commercial toolbars, add-ons are often designed as small applications that run in the browser, capturing browser events and

supplemental data in forms, for example. Add-ons may also be used as one component in an architecture, for example, in coordination with a proxy server.

Custom browsers. Alternatively, client-side data may be collected using a custom browser, which may operate as a single, integrated application for logging, control of experimental factors, and collection of supplemental data.

Injected client-side script. Where a study requires client-side data for users interaction with only a single server, client-side logs may be generated on a page-by-page basis. In this case, scripts that invoke logging functionality (e.g., event detection, recording, and transmission to the server) may be inserted into pages before they are sent to the user.

System-wide logging applications. A stand-alone logging application may be used to record interaction. Fenstermacher (2003) proposed a system with access to events occurring across the operating system of users' machines, so that interaction with every application and process could be recorded; there is no evidence that the system was ever deployed. Kelley (2005) used a commercial logging system that recorded events across all applications on the machine.

2.1.c. Control method

Proxy-based controls. A proxy server may be used as a component in a client-side data collection system. Generally, in HCIR research this is done for experimental manipulation. The proxy may simply log clickstreams, it may log additional data such as webpage html, or it may intercept messages and pages. Manipulations can be used to change intercepted outbound requests, or to alter intercepted inbound pages, all without the explicit knowledge of research subjects. A proxy may be invoked using an add-on or a custom browser.

Website controls. In client-side logging, website controls may have several purposes. For completely remote studies, they may control download, installation, and activation of an add-on. They may also be used to control experimental factors and tasks, and to collect supplemental information, such as demographics and responses to questionnaires. KLRT provides the option for website controls.

2.2. Research designs

The systems described above have been deployed in support of a broad range of research designs (see Appendix 1, Table 2). The design of a study may affect subject recruiting and retention. A potential research subject's decision to participate is affected by the place in which the study occurs, and the means required for using the experimental system. Once a subject has been recruited, subject retention is affected by the types of tasks encountered during the study, and the obtrusiveness of the observations. We review these factors as they relate to the design of the system.

Installation procedure. Client-side logging software may run on a machine in a laboratory, or it may run on research subjects' own machines. Here we report on systems designed primarily for deployment on a subject's own machine. In studies of this type, a researcher may install the logging software for each subject, or, in a completely remote study subjects must download and install the software on their own. Search engine add-ons are downloaded and installed when users install associated application software such as a toolbar. Users grant permission for recording of their search activities when they "opt-in" during installation of the application software. In contrast, in academic research, potential subjects must receive a detailed and often complex IRB-approved disclosure, and then consent to participate *before*

downloading an add-on. KLRT may be used in a laboratory or in remote studies. It is designed specifically for studies requiring IRB approval.

Type of task. In both longitudinal and cross-sectional designs, search tasks may be assigned to research subjects, or subjects' own *in situ* search may be observed. *In situ* tasks may be observed unobtrusively, or subjects may be asked to provide supplemental data in coordination with their tasks. KLRT may be used for any of these types of designs.

3. KLRT COMPONENTS

3.1. Add-on.

3.1.a. Overview

The add-on has a simple toolbar interface that allows subjects to control logging and upload functions. The interface is written in *XUL*, and the add-on's functional code is written in *JavaScript*. The code can be adapted to activate or disable optional interface controls and functions. By adapting the add-on code, different types of protocols and experimental designs may be facilitated. Add-on processing is activated by the set of *browser events* listed in Table 5, and it is controlled by a set of *event handler functions*. Below we describe these components, providing details where we have added new functionality or made changes to LQLT 2.1.

3.1.a.i. Toolbar Interface.

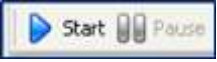



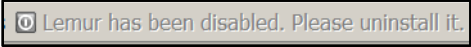
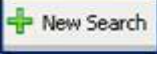




There are nine available toolbar controls in the interface. All the controls are displayed in Figure 2, however, when setting up the toolbar for a specific protocol, optional controls may be removed from the interface in order to match the requirements of a study design.

Each control is activated by simply clicking its icon. Clicks generate *browser events*, which are detected by the add-on (see section 3.1.c below). Table 1 lists each control, its function, and options for configuration. *Start/Pause* turns the logging function on and off. *View* opens a display of either the activity log (see Figure 3) or the search log (see Figure 4). *List* opens a small window to display the *whitelist* (described below). *Disable* initiates a "failsafe" mechanism that allows subjects to withdraw from the study by simply clicking the button and confirming the intent to quit. When the add-on is disabled, uploads are prevented and add-on functions cannot be restarted, even if the add-on is reinstalled. Subjects click *New Search* when starting a new search objective. This opens a small form window for data collection (see Figure 6). Values collected in the form are written to the search log. Subjects have the option to *bypass* the form. *Clear* empties all content from both the activity log and the search log, after the subject confirms the intent to clear. *Upload* starts the upload of both logs, after the subject confirms the intent to upload. *Settings* opens a small form window that allows subject to set a preferred daily upload time (see Figure 7). *Help* displays a pop-up box with contact information for help with the study.



Figure 2. Toolbar interface

Table 4. Toolbar interface

Icon	Function	Options
 4	Starts and pauses the logging function; when start button is active (blue), the pause button is grey and logging is off; when pause button is active (blue), logging is on and the start button is grey	Default mode for Firefox restart defined at configuration: can start in started, paused, deactivated, or disabled mode
 4 (see Figure 3 and 4 for examples of log displays)	Opens small drop-down menu with choice to display contents of either <i>activity log</i> or <i>search log</i> ; display opens in the active browser tab	n/a
 5 (see Figure 5 for example of <i>whitelist</i> display)	Opens a small window to display <i>whitelist</i> search system urls	Whitelist defined at configuration; whitelist not editable by user
 6 	“Failsafe” quitting mechanism allows users to quickly withdraw from the study; it disables all add-on functions and prevents uploads, restart of logging, and reinstall or activation of the add-on; a pop-up box asks user to confirm disable	n/a
 7 (see Figure 6 for form box)	Opens a pop-up box that asks users to describe search objectives; form values are written as a single string to the search log; users may “ignore” the form	Text and form fields may be changed at configuration
 4	Deletes both the “activity log” and “search log”; a box asks user to confirm clear	n/a
 4	Initiates a confirmable upload of both logs to the backend server; a pop-up box asks user to confirm upload	n/a
 8 (see Figure 7 for form box)	Allows user to set a preferred upload time; used in longitudinal designs to create a periodic upload reminder	Set upload time interval
 4	A pop-up box displays contact information for help	Text in message may be changed at configuration

⁴ included in LQLT

⁵ adapted from Russell and Oren (2009), renames the LQLT “search engine” function as “whitelist”

⁶ new function in KLRT

⁷ adapted from Fox et al. (2005), new function in KLRT

⁸ new function in KLRT replaces the LQLT “settings” function

Table 5. Logged browser events

Event name	Event description	Related functions
AddTab	add tab	log activity event
Blur	blur	log activity event
CtrlC	control c (copy)	log activity event
Focus	focus	log activity event
Hide	hide tab	log activity event
LClick	left click	log activity event
LoadBub	load bubbled	log activity event
LoadCap	load page (captured)	<ul style="list-style-type: none"> • log activity event • website control processing (3.1.c.ii.a) <ul style="list-style-type: none"> ○ update whitelist ○ start add-on ○ start non-confirmable upload ○ disable add-on • blacklist processing (3.1.c.ii.b) • whitelist processing (3.1.c.ii.c) • search session processing (3.1.c.ii.d)
MClick	middle click	log activity event
PauseLogging	pause logging function	<ul style="list-style-type: none"> • log activity event • set state to paused and reset start/pause buttons (3.1.c.iii.a)
PopUPSearchbox	open "search purpose" box	<ul style="list-style-type: none"> • log activity event • search session processing (3.1.c.ii.d) • new search (3.1.c.iv.a)
RClick	right click	log activity event
RmTab	remove tab	log activity event
ScrollBegin	begin scroll position	log activity event
ScrollEnd	end scroll position	log activity event
Search	submit query string	<ul style="list-style-type: none"> • log activity event • blacklist processing (3.1.c.ii.b) • whitelist processing (3.1.c.ii.c)
SelTab	select tab	log activity event
Show	show tab	log activity event
StartLogging	start logging function	<ul style="list-style-type: none"> • log activity event • set state to started reset start/pause buttons (3.1.c.iii.b)
ViewLog	click view log	<ul style="list-style-type: none"> • log activity event • display log in browser window (3.1.c.iii.c)

Events that trigger processing in addition to an activity log entry are in bold

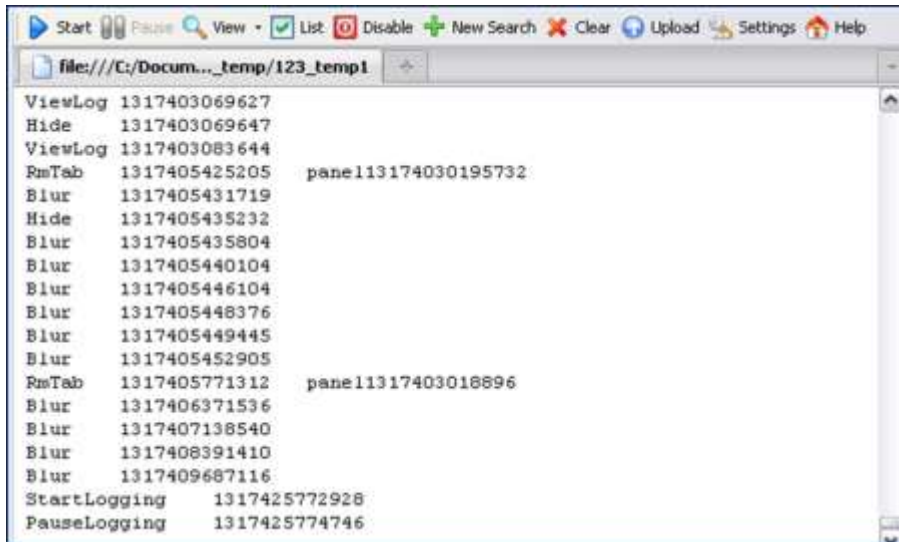


Figure 3. Activity log display for subject, accessed via *View* button

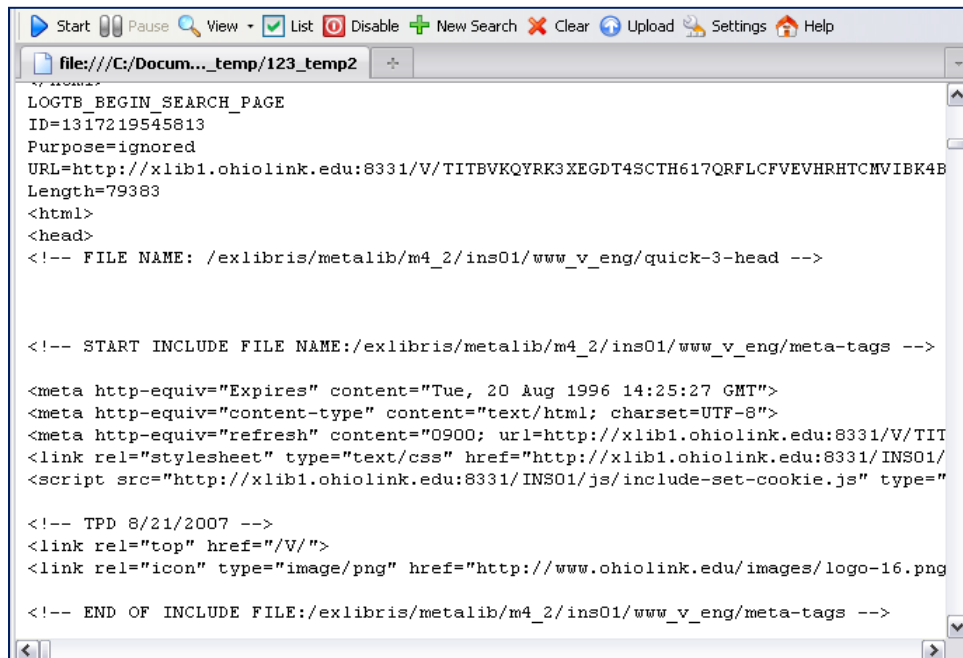


Figure 4. Search log display for subject, accessed via *View* button



Figure 5. Whitelist

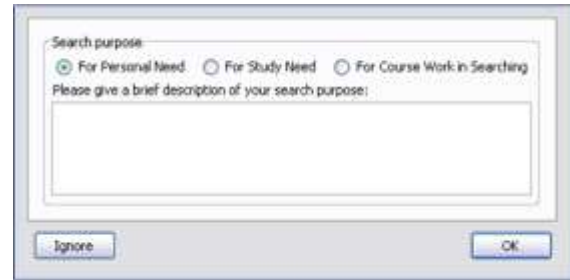


Figure 6. Pop-up form box for recording search objectives

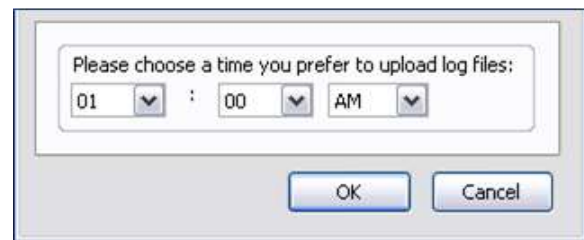


Figure 7. Pop-up form box for setting preferred daily upload

3.1.a.ii. Logging.

KLRT uses the logging functions found in LQLT 2.1, with the exception of blacklist processing, which we have modified (described below). The *activity log* (see Figure 3) records the set of FF browser events listed in Table 5. We have added logging for *view log* events, and for the display of the *new search* form. The *search log* (see Figure 4) records source code for any page loaded from the url/paths listed in the *whitelist* (see Figure 5), along with the timestamp/ID of the query request. The search log also records values collected in the new search form.

3.1.a.iii. Upload.

The KLRT uses the upload functions provided with the LQLT 2.1.

3.1.b. Add-on installation, initialization, and activation status

3.1.b.i. Installation files.

Like LQLT 2.1, when installed, KLRT creates a directory (123_temp) and two log files (activity log: 123_temp1 and search log: 123_temp2) in the user's Mozilla Firefox profile. Two parameter files (*defaults.js* and *configuration.js*) are also installed, along with other source code. The first, *defaults.js* contains the minimal default configuration for LQLT 2.1. The second, *configuration.js*, contains static values for the specific configuration of the add-on.

3.1.b.ii. Initialization.

The add-on is initialized when FF restarts after installation. Whenever FF is restarted, the add-on tries to read values from variables in the FF preferences file *prefs.js*. If the variables are not found in *prefs.js*, initial default values are read from *defaults.js* and written to *prefs.js*. As the add-on is used and initial values are changed, the updated values are stored in *prefs.js*. Some of the variables stored in *prefs.js* include, whitelist urls ("knownSearchEngines"), the add-on's status ("lemurlog_g_enable"), and the start time for the next automatic upload ("nextTimeToAutoUpload"). Variables with static values are stored in *configuration.js*, for example, the address of the backend server ("serverBaseURL"). Note that uninstalling the add-on does not erase the values stored in the *prefs.js*. In order to reinstall the add-on with new values in *defaults.js*, the user must delete the *prefs.js* file before the re-installation.

3.1.c. Event handler functions

The add-on uses two main code files for event handling (*logtoolbar.js* and *util.js*). Main processing, including event detection and major functions, are in *logtoolbar.js*. Utility functions (most from LQLT 2.1) and the website control functions are in *util.js*.

KLRT uses the event handlers found in LQLT 2.1, some of which we have revised and adapted. We have also added new functionality. The event handler listens for browser events listed in Table 5. The add-on's functions are triggered only when an event is detected. Two functions are invoked by any event: checking the add-on's activation status, and checking a timer for automatic periodic uploads (see section 3.1.c.i). For most event types, the only other function invoked is a write to the activity log, however, six event types are also associated with additional functions. Several of the event types are not standard browser events, but are generated during add-on processing. *StartLogging* and *PauseLogging* events switch the start and pause interface buttons, and the add-on's state. *ViewLog* events open the requested log file for display in the active browser window. Section 3.1.c.iii describes these three events. *LoadCap* events (a page-load in the browser) trigger several important functions (see section 3.1.c.ii). *Search* events are logged when a *LoadCap* event occurs under certain conditions. KLRT includes two new event types: *Disable* and *Searchbox*. *Disable* events change the add-on's status, and clear the log files. *Searchbox* processing controls a timer, and collects supplementary data from the user via a popup form. Section 3.1.c.iv describes these two events.

3.1.c.i. Functions triggered by all event-types

3.1.c.i.(a) Activation status check.

The KLRT add-on may be coded to install in one of three activation states: *started*, *paused*, or *deactivated* (see section 3.1.iii). The activation state can be changed by a button event or by website control processing. It is switched to a paused state when the Pause button is clicked, and to a disabled state when the user confirms a click on the Disable button. KLRT uses a status code in the user's *prefs.js* file to maintain its activation state across all instances of FF.

Activation status check is used to maintain KLRT's state across FF instances. When KLRT detects any event, the add-on's status code is always checked, as functionality is dependent on the activation state. For most event types, no action occurs if the add-on is deactivated, paused, or disabled, however, there are several functions that may be initiated even when the add-on is in these states.

3.1.c.i.(b) Periodic upload.

For studies that don't use a controlling website, only the user can trigger a log upload from the client. The periodic upload function is particularly important for studies of this type. The function allows the user to set a preferred time for a daily upload. Two variables control the procedure:

"autoUploadIntervalTime" defines the upload period. Currently the default configuration uses a 24-hour upload interval (86,400 seconds); this may be changed to any value.

"nextTimeToAutoUpload" records the unix time for the next auto-upload.

Periodic upload processing is invoked after any event is recorded in the activity log. The add-on calls the function "lemurlog_checkAutoUpload". If the current time exceeds "nextTimeToAutoUpload", a confirmable upload event is triggered and the "nextTimeToAutoUpload" variable is incremented by the "autoUploadIntervalTime". If more than 48 hours have elapsed, only one upload event is triggered. When the event is triggered, a box pops up asking the user to confirm to start the upload. Users can skip the upload by clicking a button.

3.1.c.ii. Functions triggered by page-load detection (LoadCap)

Like LQLT 2.1, KLRT listens for completion of page loads in the browser. In KLRT, we have added *website control processing*, which uses special urls to trigger add-on functions. We have disabled the blacklist interface in LQLT 2.1 and replaced its functionality with simple *blacklist processing* for determining whether a url should be logged. We use the unchanged LQLT 2.1 *SearchEngine* function for what we term *whitelist processing*. We have also added *search-session processing* that detects search sessions and collects supplemental information from users. Below we describe the processing and sequencing of these functions.

3.1.c.ii.(a) Website control processing.

KLRTK includes a website structure that may be coupled with the add-on to trigger a set of control functions (the website structure and server-side processing are detailed in section 3.2 below). Website control processing is invoked after a page-load is detected. A simple string match determines whether a *triggering url* has been detected. The website can trigger four basic functions: updating the whitelist, starting the add-on, disabling the add-on, and uploading the logs. None of these events are logged.

Update whitelist.

Over the course of a long longitudinal study, it may be necessary to change the search-string urls in the whitelist. We use a simple web interface for this purpose; users login to the website to activate the update. The whitelist is updated when a triggering url is detected. The new search-string urls are passed to the add-on by appending each to the triggering url, with each separated by '@'. When triggered, the function "lemurlog_update_whitelist" parses the triggering url to extract the new search-string urls. These are then appended to the whitelist in the *prefs.js* file, and the browser is redirected to the triggering url without the appended items.

Start add-on.

When triggered, the function "lemurlog_ToStart" puts the add-on in the "started" state, and then calls the function that resets the start/pause buttons. We use this function to activate the add-on when it is installed in the "deactivated" state (see section 3.2.b.iii, below), or to automatically start logging when a task-administration page is loaded (see section 3.2.b.iv, below).

Start non-confirmable upload.

We use this function for studies in which users are assigned search tasks through a controlling website. The upload is triggered when server-side processes confirm that the task has been completed. The upload trigger is dependent on the user entering a specific sequence of webpages, so that an upload cannot be started by requesting the triggering url. When triggered, "lemurlog_IsTask2URL" starts a log upload function that does *not* ask the user to confirm. See section 3.2.b.v, below, for a description of website processes.

Disable add-on.

We use this function in a website-controlled process that verifies that the add-on has been uninstalled. When triggered, "lemurlog_IsClearTime" disables the add-on without requesting confirmation from the user. The disable function is described in section 3.1.c.iv.b, below.

3.1.c.ii.(b) Blacklist processing.

In order to protect the privacy of research subjects, we do not log any urls with a secure connection (i.e. https connections), nor any urls containing the string "mail" (adapted from Russell & Oren, 2009). Such a url is termed *nonrecordable*. There is one exception to the secure connection rule. If the url is "https://www.google.com/search?" it will be deemed *recordable*. This is required because users logged into Google accounts receive all search result pages via a secure connection.

Blacklist processing is invoked after a url is processed by the website control handlers; urls are examined by a function called "lemurlog_IsRecordableURL", which uses a string match to check the url for "https" and "mail". If the url is recordable, its value and the "LoadCap" event are written to the activity log. After the record is written, whitelist processing is called.

The current blacklist code contains a function called "washAndRinse", which originated in LQLT 2.1. This function has been modified in KLRT, but it is currently disabled. See section 4.2 for a discussion of future work on this function.

3.1.c.ii.(c) Whitelist processing.

KLRT whitelist processing uses the unchanged LQLT 2.1 "SearchEngine" processing to detect and log search engine results pages, however, unlike LQLT 2.1, users are unable to alter the list of search systems displayed in the interface. Whitelist urls are hardcoded in the KLRT configuration file. After a "LoadCap" event is recorded, the add-on function "lemurlog_IsSearchURL" reads the prefixes of whitelist urls from the LemurLogToolbarConfiguration structure, and compares them with the current url. If the current url matches at least one whitelist url, the function returns true. The Search event is then recorded in the activity log. If search-session processing is active

in the KLRT code, processing continues there (see next section). If not, the html of the search results page is immediately recorded in the search log.

3.1.c.ii.(d) Search-session processing.

For studies of *in situ* search, where there are no search task assignments, it may be helpful to record information about a user's naturally occurring search objectives, or other desired information. Activities associated with a single search objective constitute a *search session*. Generally, if a user doesn't submit a new query for a period of 30 minutes or longer, it is likely that the next query entered will be associated with a new search objective. In log analysis, a 30-minute interval is used to demarcate search sessions. We use the 30-minute heuristic to trigger a popup box that asks users for information about their search objectives.

Search session boundaries are detected based on a timing variable, "_lasttime_purpose"; the value is initially zero, and it is updated with the time of the first search query submitted after the add-on initializes. Every time whitelist processing returns true, "_lasttime_purpose" is compared with the current unix time. If 30 minutes have elapsed, the *search session* form (see Figure 6) pops up, asking the user for information. Every time whitelist processing returns true, "_lasttime_purpose" is updated with the current time.

Values submitted in the search session form are stored as a single string in the variable "_search_purpose". Immediately before the html of the search results page is recorded in the search log, the "_search_purpose" string is recorded in the search log.

3.1.c.iii. Add-on events that trigger LQLT 2.1 functions

3.1.c.iii.(a) Pause (PauseLogging event)

When the "Pause" button is clicked, the state variable in prefs.js is set to *paused*, the start/pause buttons are reset, and the PauseLogging event is recorded in the activity log.

3.1.c.iii.(b) Start (StartLogging event)

When the "Start" button is clicked, the state variable in prefs.js is set to *started*, the start/pause buttons are reset, and the StartLogging event is recorded in the activity log.

3.1.c.iii.(c) View (ViewLog event)

When the "View" button is clicked, a two-line menu drops down and the user may select the activity log or the search log. When a log is selected, the associated log file is displayed in the active browser tab and the ViewLog event is recorded in the activity log.

3.1.c.iv. Add-on events that trigger KLRT functions

3.1.c.iv.(a) New Search (Searchbox event processing)

When the New Search button is clicked, the search session form pops up (see Figure 6). The user has the option to complete the form and submit it, or ignore it. The timing variable "_lasttime_purpose" is updated with the current time, and the string variable "_search_purpose" is updated. Search session processing is invoked the next time a whitelist even occurs (see section 3.1.c.ii.c above).

3.1.c.iv.(b) Disable (Disable event processing)

We added the Disable button to the toolbar to allow users to quit a research study by clicking a single button. When a disable event is triggered, users are asked to confirm that they want to disable the add-on. If the answer is positive, then the add-on deletes both log files, a file called "disable" is created in the user's FF profile, and the add-on's status is set as *paused* in *prefs.js*. As shown in figure 8, the Start, Pause, Disable, New Search, Upload, and Settings buttons are then greyed out and a disabled message is displayed. The add-on will no longer record any activity in either log, and it cannot be restarted, even if it is reinstalled. The only way to reactivate the add-on is by deleting the "disable" file before the installation.

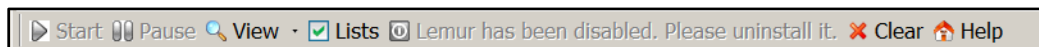


Figure 8. Toolbar interface after disable

3.2. Website.

The KLRT control website has four components (see Figure 1, above), which are used in sequence. The website can be configured for remote studies, where a subject downloads and installs the add-on, or for *in-lab* studies, where the add-on is installed on machines in the lab. The website runs php with a MySQL database, which contains information about subjects, the structure of the study, and the subject's status within the sequence of activities that define the study protocol.

3.2.a. Website sections

3.2.a.i. Section 1 – instructions for installing the add-on

The first section is used only for remote studies. The primary purpose of the section is to insure the correct machine environment for the study, and the correct installation process for the add-on. Instructions for setting up the environment (e.g., instructions for turning Google Instant off) and downloading and installing the add-on, are presented. Users must login to the study website in order to reach the add-on download (L_1).

3.2.a.ii. Section 2 – instructions for assigned tasks

This section is used for any study with assigned tasks that must be completed via the website. For studies conducted in the lab, where the user does not need to install the add-on, users enter the study website by logging in at section 2 (L_2). After login, the website may trigger validation of the add-on installation (A). A timer may be used to test the elapsed time between the start of an add-on download and the start of installation validation (T_1). If the test occurs after the deadline, a control action can occur. For example, the user may be excluded from the study, or the user may be required to download a new version of the add-on.

3.2.a.iii. Section 3 – task interface and administration

The third section controls the study protocol by administering task assignments and collecting data via forms. The section should be customized to meet the requirements of the study (e.g., number of tasks, task assignment method, time limits, form text and variables, and database connectivity). The section is designed to use a repeating task structure, where a sequence of pages may be used iteratively, according to the protocol structure. There are three basic functional pages.

Task-administration. The task-administration page displays any values or forms required by the active task, which may be assigned from the protocol areas of the website database. For

protocols that use a time-limit (T_3), the page may also display a count-down stopwatch. Upon entering section 3, the page may trigger activation of the add-on (B) or a forced start of logging (C). Form submission from this page may request a confirmation page.

Task confirmation. The confirmation page allows the subject to confirm the upload, or quit the study without an upload. If confirmed, the page starts the log-upload and posts the form data to the website database. For example, this can be used to upload logs simultaneously with a form submission. If the subject quits, disable processing is initiated (see section 3.1.c.iv.b).

Log upload. This page runs a non-confirmable upload of the logs (D). It also may be used to activate the process that disables the add-on (E). For example, this could be used to disable the add-on after the last task is completed.

3.2.a.iv. Section 4 – instructions for uninstalling the add-on

The fourth section of the site provides any instructions for the end of the study, and for uninstalling the add-on. For studies that don't use assigned tasks, such as a longitudinal study of *in situ* search, the user may sign in to this section without ever using the other sections (L_3). The section tests for an active installation of the add-on (F), with the objective of assuring that the add-on is disabled after the study is completed.

3.2.b. KLRT control functions on the website

3.2.b.i. Identify study subjects at login (L)

When users login on the study website, after password verification, the user's subjectID is read from the website database. A web page is then returned to the user, and the url of that page contains a special "signature#" string, followed by an encrypted subjectID. The add-on recognizes the signature string, and captures the encrypted subjectID, which is passed to the backend during upload, and recorded as "ClientSessionID" in the "tblClients" table of the backend database (see below). Also, the identification process checks for restart status (see section 3.2.c.ii).

3.2.b.ii. Validate add-on installation (A)

We want to prevent subjects from completing assigned tasks without a working installation of the add-on. Also, in some designs it is essential to prevent the user from installing the add-on while certain pages of the website are open. To prevent these problems, the website can test for a valid installation of KLRT. The procedure uses a sequence of two triggering urls.

The first url triggers the "start logging" function (see section 3.1.c.iii above). It is appended with a unique signature: an encrypted combination of the user's "subject ID" and the current time stamp. The whole url is recorded in the activity log.

The second url triggers a "start non-confirmable upload" (see section 3.1.c.ii.a). Once the upload is complete, the back-end parses the activity log and updates the backend database.

In the third step, the backend database is interrogated to determine whether the unique signature is found. When the signature is found, a valid add-on installation is confirmed, the user is notified, and website processing continues. If the signature is not found, the website redirects to the installation instructions and issues a message to the user. Progress through the control website cannot continue until the installation is validated.

3.2.b.iii. Activate add-on (B)

The add-on may be coded so that it installs in a "deactivated" state. When "deactivated", both the start and pause buttons do not work. Users cannot start the add-on until they sign in to the controlling website using their user names and passwords, which are available only to consenting research subjects. Once a user is logged in, a trigger page for "start add-on" is sent to the client and all buttons are activated (see section 3.1.c.ii.a above).

We use this function for several purposes. First, it forces users to encounter the installation instructions before the add-on can be activated. Our goal is to reduce the chance that a user might inadvertently log and upload private information before learning about how the add-on works. Also, the function provides us with confirmation that the user has installed the add-on. This is particularly important for studies that use an unconfirmed upload trigger (see below).

3.2.b.iv. Trigger add-on start and log clear (C)

In order to guarantee that the add-on is active at the beginning of each experimental task, the task component of the website may control add-on status. In some cases, we want the add-on to be on at the beginning of each search task, and the urls of task-related web pages can be set as *start add-on* triggers (see section 3.1.c.ii.a, above). In studies where task assignments take more time, perhaps weeks, we want to minimize the risk that the add-on will be active when it is not needed, and that unnecessary log files will be uploaded. In this case, we set the add-on to start in a deactivated status, so that when any Firefox instance is initialized, the add-on starts in this status. The add-on is activated only when the subject successfully logs into the web site, which triggers the add-on to clear the log files and then start logging.

3.2.b.v. Trigger log upload (D)

In some experimental studies, we want log data to be uploaded as soon as a task is finished, so that separate log files are created for each task. In this case, task pages are set to trigger a *non-confirmable upload* (see section 3.1.c.ii.a, above). The triggering page uses a special "?submit" string, which is not visible to the user, and which is appended only when the page is entered directly from a prior controlling page. This prevents an uncontrolled, non-confirmable upload if a user restarts a task sequence by reloading a triggering page.

3.2.b.vi. Trigger log clear (E)

In most studies, we want the add-on to be disabled after all of the assigned tasks are complete. In this case, the url of the last page in a sequence of task pages can be set to trigger the *add-on disable* function (see section 3.1.c.ii.a, above).

3.2.b.vii. Validate add-on uninstall (F)

At the conclusion of a study, it may be important to verify that that add-on has been removed from a user's system. The procedure for this is very similar to the "validation of add-on installation" above (see section 3.2.b.ii). The only difference is the processing that occurs in the third step. When the signature is not found, a valid add-on uninstall is confirmed, the user is notified, and website processing continues. If the signature is found, the website redirects to the uninstall instructions and issues a message to the user. Progress through the control website cannot continue until the uninstall is validated.

3.2.c. Other control functions

3.2.c.i. Monitor installation timers (T)

In order to minimize the risk that the add-on is downloaded and installed at some distant time, or by someone using the machine but unfamiliar with the study, we may use timers in the website. In some cases, we simply want to know that the add-on has been installed immediately after download. In this case, the elapsed time since download is checked when the installation is validated (T_1). In other cases we want subjects to download the add-on immediately before completing assigned tasks, and the timer checks the elapsed time between download and the start of the task section of the website (T_2). In both cases, when the download starts, a timestamp is recorded in the "lemur_download" field in the "subject" table of the website database. This field is checked by the controlling webpage, and if the allowed time has elapsed, the subject is eliminated from the study, and the browser is redirected to the directions for uninstalling the add-on.

3.2.c.ii. Restart incomplete task-set

During a task-based protocol, subjects may close the browser tab or window after starting a task-set, but before completing the set. If the browser attempts to open the task-administration page, but has not saved the state for restart, a page reload is requested from the server. The task-administration page tests for an active session as an indication that the subject is logged in. If no session is found, a redirect serves the login page. Upon login, the subject is identified (see section 3.2.b.i) and the identifier for the last active task is retrieved. If the protocol uses a time limit for task completion (T_3), we also read the value for the remaining-time. These status values are stored in the session variable and additional task-related data is retrieved from the database. Unless the time limit has been reached, the task-administration page is populated with task-related values and the page is served. The task-administration page passes the client's unique signature, and it may also be used to trigger a *start add-on* (see section 3.1.c.ii.a).

3.3. Backend

KLRT uses the backend of LQLT 2.1, however, we have made several modifications. The backend has three main components: one handles information about clients sending logs to the server, and two others handle parsing and storage of the activity and search logs. We discuss these below. The backend uses a MySQL database. Incoming logs are stored in temporary files while backend processing is performed.

Each log upload creates two temporary files, an activity log and a search log. During upload, the backend extracts the data from the upload stream, including an encrypted unique identifier for the client. The identifier is stored in the backend database in the field named ClientSessionID (see below). The identifier may be associated with a research subject (see section 3.2.b.i) or a machine. The ClientSessionID may be hardcoded in the add-on prior to installation.

Because we use KLRT to record search activities in a large number of diverse search systems, our search results pages have many various formats. While LQLT 2.1 has parsers for several formats (Google, Bing, Yahoo, MSN, and Sogou), we obtain relatively few samples for many of the formats we capture. Because we don't have parsers for these formats, we rename and store all unparsed temporary log files as described below. For large studies involving more than one user at a time (i.e. studies not conducted one-on-one in the lab), we have turned off the parsing functions on the backend (see section 4.7 below).

3.3.a. Client information component.

This component runs first, and records the identity of the source of each log. Most of the code for this component is in a file called *ClientUploadInformation.java*. It captures the ClientSessionID, IP address, and browser type from the upload stream, and then compares this information with data in the "tblClient" table to look for an associated ClientID. If a ClientID is not found, a new ClientID is assigned in the table. The ClientID is then associated with a new LogID, which is assigned sequentially in the "tblLog" table.

3.3.b. Activity log component.

The second component runs next. It parses and stores incoming activity logs. Most of its code is in a file called "ActivityLogProcessingThread.java". First, it reads the event types from table "tblLogActionType" (see Table 2 above). It then parses the activity log and inserts corresponding information into tables such as "tblLogAction" and "tblLogPages" based on the event types and previous data in the tables. In KLRT we modified the activity log component so that it saves the raw activity log. When the parsing is finished, the temporary activity log is saved in a folder called "logfiles", and it is renamed "ActivityLog" followed by the logID that has been assigned by the first component. The temporary activity log is then deleted. In this component, we added new event types to "tblLogActionType" and made modifications to the parser for scrolling data.

3.3.c. Search log component.

After the second component is done, the third component runs. Its code is in a file called "SearchLogProcessingThread.java". This component parses the search log and inserts corresponding information into tables such as "tblSearchResults" and "tblSearchPages". The information includes the ID of the search page (note that this is not the "SearchPageID" field in "tblSearchPages" table, but the panel ID), the url address of the search page, and the length of source code of the search page. In addition, parsers for Google, Bing, Yahoo, MSN, and Sogou produce detailed information for each results page, such as each item, the rank of each item, and so on. In KLRT we modified the search log component so that it saves the raw search log. After finishing the parsing, this component copies the temporary search log to the "logfiles" folder and renames the file "SearchLog" followed by the logID assigned by component one. Then it deletes the temporary search log. We modified the code in this component so that it parses and stores fields captured in the search purpose popup box. We also modified the code so that raw search results pages are not stored in a table, as is done in LQLT 2.1.

4. CHALLENGES AND FUTURE WORK

While the current version of KLRT meets many of our objectives, several major challenges remain, and we have the inevitable list of enhancements we have not yet implemented. We discuss these below.

4.1. AJAX stream capture.

Among our biggest challenges, the greatest is the need for a process that can listen to an AJAX stream and capture the final resultant html display. This issue is most obvious in the AJAX interface functionality of Google Instant. Currently, if Google Instant is on, KLRT cannot capture Google query strings and search results pages. The problem extends, however, to other search systems such as Dialog. We expect that AJAX will become more common in search systems, including library and database systems. For KLRT to continue to be useful this problem must be solved.

In order to get around this problem, before subjects install the add-on we instruct them to turn Google Instant off. In pilot testing, we found that most people were not aware of Google Instant, even if it was

on. Generally, they were also unfamiliar with changing settings for Google. We monitor uploaded data to check for evidence of search activity where there are no search urls or results pages in the logs and send reminders to turn Google Instant off.

4.2. Smarter blacklist processing – keyword-based filters

KLRT lacks a robust and efficient user-controlled blacklist process. The blacklist process in LQLT 2.1 provides a pop-up window that allows users to specify private information that should not be recorded in the logs. LQLT 2.1 blacklist processing scans whitelist urls for blacklist values, and when they are found, processing attempts to overwrite the values before the url is logged. Our specifications for blacklist processing are different; we require that the user be able to exclude an entire url from logging if blacklist values are found in the url. This is akin to blacklist processing currently in KLRT, in which we exclude urls containing the strings “mail” or “https”.

In the current version of KLRT, we have disabled the blacklist interface, as we have several problems that remain to be solved. We have not yet developed a reliable process for detecting blacklist stings in urls. This requires a more sophisticated JavaScript string handler. Also, a single version of the current blacklist values must be available to every instance of FF in time for blacklist processing; this is not supported in LQLT 2.1, so a user’s changes to the blacklist are not applied to other browser instances that are active at the time of the change. Meeting this requirement requires modification to the processes for maintaining state and to the processing flow triggered by LoadCap.

4.3. Extracting results lists from pages using frames

The current version of KLRT does not have a process for extracting html from search results pages that use frames. We plan to address this need on a system-by-system basis.

4.4. Parsers for library and database search systems

The current whitelist contains over 170 search systems. We are in the process of developing parsers for systems logged with highest frequencies. We plan to make the parsers available as they are tested.

4.5. Tab and copy event processing for Firefox 4.0+

The current version of the add-on is designed to work on FF 3.6. While it is possible to install and run the current version on FF 4.0 or greater, *tab* and *copy* events are not logged. Implementation of the fix is forthcoming.

4.6. A modular, configurable version of the toolkit

Ideally, KLRT should be truly configurable, so that researchers with minimal programming resources can implement various study designs. This enhancement would require development of configurable code, and an interface for setting configuration parameters for the add-on and the website.

4.7. Modify backend for multi-thread update

The LQLT 2.1 backend does not support mutli-thread processes. For studies where multiple users submit simultaneous log uploads, this functionality is needed to keep the backend running. For larger studies, the current work-around is to turn off the log parsers, and simply record log-uploads and rename the saved raw logs for parsing and processing at a later time.

5. References

- Atterer, R., Wnuk, M., & Schmidt, A. (2006). Knowing the user's every move: user activity tracking for website usability evaluation and implicit interaction. *Paper presented at the Proceedings of the 15th International Conference on World Wide Web*, Edinburgh, Scotland.
- Bierig, R., Gwizdka, J. & Cole, M. (2009). A User-centered experiment and logging framework for interactive information retrieval. *Paper presented at the Workshop on Understanding the User at the 32nd Annual ACM SIGIR Conference on Research and Development on Information Retrieval*, Boston, MA.
- Baker, L. (2004). Google Partners with RealPlayer for toolbar distribution. *Search Engine Journal*. Retrieved from <http://www.searchenginejournal.com/google-partners-with-realplayer-for-toolbar-distribution/467/>
- Capra, R. (2009). HCI Browser: a tool for studying web search behavior. *Paper presented at the 73rd ASIS&T Annual Meeting on Navigating Streams in an Information Ecosystem*, Pittsburgh, Pennsylvania.
- Cartright, M.-A., White, R. W., & Horvitz, E. (2011). Intentions and attention in exploratory health search. *Paper presented at the 34th Annual International ACM SIGIR Conference on Research and Development on Information Retrieval*, Beijing, China.
- Choo, C. W., Detlor, B., & Turnbull, D. (2000). Information seeking on the Web: An integrated model of browsing and searching. *First Monday*, 5(2).
- Claypool, M., Le, P., Wased, M., & Brown, D. (2001). Implicit interest indicators. *Paper presented at the Proceedings of the 6th international conference on Intelligent user interfaces*, Santa Fe, New Mexico.
- Downey, D., Dumais, S., & Horvitz, E. (2007). Models of searching and browsing: languages, studies, and applications. *Paper presented at the Proceedings of the 20th international joint conference on Artificial intelligence*, Hyderabad, India.
- Downey, D., Dumais, S., T, Liebling, D., & Horvitz, E. (2008). Understanding the relationship between searchers' queries and information goals. *Paper presented at the 17th Annual International ACM SIGIR Conference on Research and Development on Information Retrieval*, Napa Valley, California, USA.
- Feild, H., A, Allan, J., & Jones, R. (2010). Predicting searcher frustration. *Paper presented at the 33rd Annual International ACM SIGIR Conference on Research and Development on Information Retrieval*, Geneva, Switzerland.
- Feild, H. A., Allan, J., & Glatt, J. (2011). CrowdLogging: distributed, private, and anonymous search logging. *Paper presented at the 34th Annual International ACM SIGIR Conference on Research and Development on Information Retrieval*, Beijing, China.
- Fenstermacher, K. D., & Ginsburg, M. (2003). Client-side monitoring for Web mining. *Journal of the American Society for Information Science and Technology*, 54(7), 625-637.
- Fox, S., Karnawat, K., Mydland, M., Dumais, S., & White, T. (2005). Evaluating implicit measures to improve web search. *ACM Transactions on Information Systems*, 23(2), 147-168.

Guo, Q., White, R. W., Zhang, Y., Anderson, B., & Dumais, S., T. (2011). Why searchers switch: understanding and predicting engine switching rationales. *Paper presented at the 34th Annual International ACM SIGIR Conference on Research and Development on Information Retrieval*, Beijing, China.

Huang, J. (2011). On the Value of Page-Level Interactions in Web Search. *5th Workshop on Human-Computer Interaction and Information Retrieval (HCIR '11)*, Mountain View, CA.

Jansen, B. J., Ramadoss, R., Zhang, M., & Zang, N. (2006). Wrapper: An application for evaluating exploratory searching outside of the lab. *Paper presented at the Workshop on Evaluating Exploratory Search Systems at the 29th Annual International ACM SIGIR Conference on Research and Development on Information Retrieval*, Seattle, WA.

Kellar, M., Watters, C., & Shepherd, M. (2007). A field study characterizing Web-based information-seeking tasks. *Journal of the American Society for Information Science and Technology*, 58(7), 999-1018.

Kelly, D. (2006). Measuring online information seeking context, Part 1: Background and method. *Journal of the American Society for Information Science and Technology*, 57(13), 1729-1739.

Kumar, R., & Tomkins, A. (2010). A characterization of online browsing behavior. *Paper presented at the Proceedings of the 19th international conference on World wide web*, Raleigh, North Carolina.

Lemur (2010). Community Query Log Project Results. Retrieved from <http://lemurstudy.cs.umass.edu/>

Matthijs, N., & Radlinski, F. (2011). Personalizing web search using long term browsing history. *Paper presented at the Fourth ACM international conference on Web search and data mining*, Hong Kong, China.

Reeder, R. W., Pirolli, P., & Card, S. K. (2001). WebEyeMapper and WebLogger: tools for analyzing eye tracking data collected in web-use studies. *Paper presented at the CHI '01 extended abstracts on Human factors in computing systems*, Seattle, Washington.

Russell, D. M., & Grimes, C. (2007, Jan. 2007). Assigned tasks are not the same as self-chosen Web search tasks. *Paper presented at the 40th Hawaii International Conference on System Sciences*, Big Island, HI.

Russell, D. M., & Oren, M. (2009). Retrospective cued recall: A method for accurately recalling previous user behaviors. *Paper presented at the 42nd Hawaii International Conference on System Sciences*, Big Island, HI.

Singla, A., White, R., & Huang, J. (2010). Studying trailfinding algorithms for enhanced web search. *Paper presented at the Proceedings of the 33rd Annual International ACM SIGIR Conference on Research and Development on Information Retrieval*, Geneva, Switzerland.

Singer, G., Norbistrath, U., Vainikko, E., Kikkas, H., & Lewandowski, D. (2011). Search-logger analyzing exploratory search tasks. *Paper presented at the 2011 ACM Symposium on Applied Computing*, TaiChung, Taiwan.

Smith, C.L. (2011). Conditions of trust for completely-remote methods: A proposal for collaboration. *5th Workshop on Human-Computer Interaction and Information Retrieval (HCIR '11)*, Mountain View, CA.

Toms, E. G., Freund, L., & Li, C. (2004). WiIRE: the Web interactive information retrieval experimentation system prototype. *Information Processing and Management*, 40(4), 655-675. doi: 10.1016/j.ipm.2003.08.006

White, R. W., & Drucker, S. M. (2007). Investigating behavioral variability in web search. *Paper presented at the Proceedings of the 16th international conference on World Wide Web*, Banff, Alberta, Canada.

White, R. W., Dumais, S. T., & Teevan, J. (2009). Characterizing the influence of domain expertise on web search behavior. *Paper presented at the Proceedings of the Second ACM International Conference on Web Search and Data Mining*, Barcelona, Spain.

White, R. W., & Morris, D. (2007). Investigating the querying and browsing behavior of advanced search engine users. *Paper presented at the Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, Amsterdam, The Netherlands.

Appendix 1.

Table 1. Studies Using Client-side Logging (includes Technical Reports)

#	authors	year	# subjects	time period	installation	type of log	type of task	privacy provisions
1	Atterer, Wnuk, & Schmidt ⁹	2006	12	lab session	in lab	browser events	assigned	consent to proxy connection
2	Capra	2009	n/a	n/a	in lab	browser events	n/a	n/a
3	Cartright, White & Horvitz	2011	660,000	6 months	commercial toolbar	clickstream	<i>in situ</i>	no <i>https</i> or intranet recorded
4	Choo & Turnbull	2000	34	2 weeks	by researcher, subject machine	browser events	<i>in situ</i>	logging on/off, view log (see Turnbull)
5	Claypool, et al.	2001	75	11 days	in lab	browser events	<i>in situ</i>	none described
6	Downey, Dumais, & Horvitz	2007	250,000	3 weeks	commercial toolbar	clickstream	<i>in situ</i>	none described other than consent
7	Downey, Dumais, Liebling & Horvitz	2008	206,000	2 weeks	commercial toolbar	clickstream	<i>in situ</i>	no <i>https</i> or intranet recorded
8	Feild, Allan, Glatt	2011	≈30	2 weeks	by subject on own machine	browser events	<i>in situ</i>	turn log on/off, view log, delete log, anonymization, upload approval
9	Feild, Allan, Jones	2010	30	lab session	in lab	browser events	assigned	n/a
10	Fenstermacher & Ginsburg ¹⁰	2003	n/a	n/a	n/a	application events	<i>in situ</i>	n/a
11	Fox, et al.	2005	146	6 weeks	by subject on work machine	browser events	<i>in situ</i>	turn log on/off
12	Guo, et al.	2011	216	4 weeks	by subject on own machine	browser events	<i>in situ</i>	no <i>https</i> or intranet recorded
13	Huang ⁶	2011	n/a	n/a	on server of target system	browser events	<i>in situ</i>	n/a
14	Jansen, et al.	2006	4	7 days	not specified	browser events	<i>in situ</i>	explicit activation of logging

⁹ proof-of-concept study

¹⁰ proposed system

#	authors	year	# subjects	time period	installation	type of log	type of task	privacy provisions
15	Kellar, Watters, & Shepard	2007	21	1 week	by researcher, subject machine	browser events	<i>in situ</i>	log entry review and deletion
16	Kelly	2006	7	14 weeks	by researcher on study machine	application events	<i>in situ</i>	subjects used laptop supplied by researcher
17	Kumar & Tomkins	2010		1 week	commercial toolbar	click stream	<i>in situ</i>	none described other than consent
18	Lemur Project	2009	n/a	12 months	by subject on own machine	browser events	<i>in situ</i>	turn log on/off, whitelist, blacklist, view log, delete log, control of log upload
19 _A	Matthijs & Radlinski (phase 1, sect. 3)	2011	50	3 months	by subject on own machine	click stream	<i>in situ</i>	random unique identifier, no <i>https</i> , dynamic pages excluded
19 _B	Matthijs & Radlinski (phase 3, sect. 6)	2011	41	2 months	by subject on own machine	click stream	<i>in situ</i>	random unique identifier
20	Reeder, Pirolli & Card	2001	n/a	n/a	in lab	browser events	<i>n/a</i>	<i>n/a</i>
21	Russell & Grimes	2007	401	2 weeks	not specified	browser events	assigned tasks	no <i>https</i>
22 _A	Russell & Oren ¹¹	2009	12	1 month	by researcher, subject machine	browser events	<i>in situ</i>	no <i>https</i> , log entry review and edit, screen capture on/off, url blacklist edit, whitelist of captured urls, control of log upload
22 _B	Russell & Oren ⁷	2009	8	1 week	by researcher, subject machine	browser events	<i>in situ</i>	same as # 11
23	Singer, et al.	2011	10	4 weeks	by subject on own machine	browser events	assigned tasks	pause/start task/logging
24	Singla, White, & Huang	2010	millions	9 months	commercial toolbar	clickstream	<i>in situ</i>	no <i>https</i> or intranet recorded
25	Toms, Freund & Li	2004	24	lab session	in lab	queries and page-saves	assigned tasks	<i>n/a</i>
26	White & Drucker	2007	2,527	5 months	commercial toolbar	clickstream	<i>in situ</i>	no identifiers assigned
27	White, Dumais & Teevan	2009	270,000	3 months	commercial toolbar	clickstream	<i>in situ</i>	no <i>https</i> or intranet recorded
28	White & Morris	2007	188,000	13 weeks	commercial toolbar	clickstream	<i>in situ</i>	none described other than consent

¹¹ study of usage of personalized homepages

Appendix 1.

Table 2. System Components

source of supplemental data	data collected				
	clicks and interaction, collected via:				clickstream via browser add-on w/out interaction data
	system-wide application	injected client-side script	custom browser	browser add-on	
proxy server	[16] a, plus html of requested pages				[19 _A] a, plus html of requested pages [19 _B] a,
website (with proxy)					[25] a
website (no proxy)				[8]	
integrated/application	[10]		[5] page evaluations [15] c, task categories and descriptions [21] a, task descriptions	[2]* [9] b, sensors, feedback ¹² [11] a, task descriptions and evaluation ¹³ [12] a, switching reasons [22 _A] b, screen shots [22 _B] b, screen shots [23] a, task descriptions ¹⁴	
no other data		[1] a [8] a * [13] a *		[4] ¹⁵ c [14] a [18] b [20] ¹¹ c	[3] a [6] a [7] a [17] a [24] a [26] a [27] a [28] a

a – collected by write to server
 b – collected by batch upload
 c – collected by researcher from machine
 * proposed system or no study reported

¹² for each task: expectations, satisfaction, frustration, evaluation; physical sense data
¹³ explicit evaluation for each search objective, and for each page visited from a SERP
¹⁴ pre-experiment demographics, and for each assigned task, pre- and post-search questionnaires
¹⁵ early client-side logging systems ran alongside the browser to detect browser events

Appendix 1.

Table 3. Study Design and System Deployment

Tasks studied:	Logging software installed:				
	<i>on subject's own machine ...</i>			<i>in lab</i>	<i>other</i>
	<i>...with commercial tool-bar</i>	<i>...by research subject</i>	<i>...by researcher</i>		
<i>assigned tasks</i>		[23] task descriptions		[1]** [9] [25]	[21] website script
<i>in situ w/supplemental data</i>		[11] [12]	[5] [15] [16]		
<i>in situ unobtrusive</i>	[3] [6] [7] [17] [24] [26] [27] [28]	[8] [18] [19 _A] [19 _B] [22 _A]	[4] [22 _B]		[14] not described

** proof-of-concept study – proxy may run on a website server, or users may point their browsers to an in-stream proxy