NIST Special Publication 500-247
January 2001

Revised: January 2002, March 2002

The FLUD format: Logging Usability Data from Web-based Applications

John Cugini / cuz@nist.gov
Information Technology Laboratory
National Institute of Standards and Technology (NIST)
Gaithersburg, MD 20899

Abstract

This paper presents a proposed format for representing the behavior of users as they interact with a Web-based application. The captured log data can be valuable for analyzing and improving the usability of such applications. The background and motivation of this effort are briefly described. The detailed syntax and semantics of the format are then defined.

Keywords

logging data; usability testing; web-based applications;

1. Background and Motivation

For the past three years, the Information Technology Laboratory of the National Institute of Standards and Technology (NIST) has been engaged in a project called NIST Web Metrics whose objective is to develop tools and techniques to support the evaluation of the usability of web sites. One of our areas of interest is recording the behavior of users as they attempt to perform given tasks during interaction with a Web-based application. The captured log data can be valuable for analyzing and improving the usability of such applications.

It soon became evident this log data is quite complex and that a common file format is needed to allow various software components (such as recorders, parsers, analyzers, and visualizers) to exchange information. The goal then is to establish a Framework for Logging Usability Data - hence: "FLUD".

2. Design Issues

2.1 Event Model

The FLUD format embodies a mid-level abstraction of user activity. Behavior such as clicking a mouse button, filling in a text entry, jumping to a new page, and toggling a checkbox is represented directly. The intention is that the FLUD model be general enough to encompass events reported from typical browsers, such as Netscape and Internet Explorer.

2.2 Scope

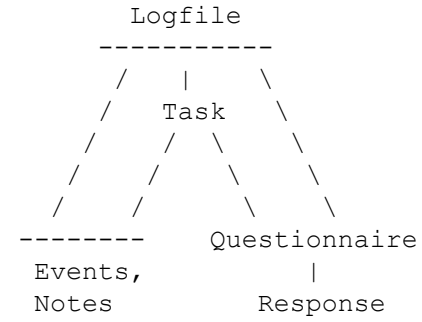
Although targeted at web applications, this format encompasses user interactions with other levels of the system. In particular, common operations performed by the browser and window system are defined, as well as those performed directly within a webpage. To the user and usability engineer, the former can be just as significant as the latter; their capture, however, may pose a challenge to generators, but if such interaction can be captured, this format provides a standard way for that information to be reported.

It is important to note that we are defining a *file format only* - the syntactic elements and their meaning. We are not in any way specifying the operation of the software components that may read or write FLUD files. Note also that this specification defines *how* an event is represented, but does not thereby require that all or any such events actually be detected and reported.

This first version of FLUD does not provide a direct representation of multiple frames within a window. We are studying how best to incorporate frames in later versions. See the description of Window_ID under Events for a recommendation on how to treat frames.

2.3 Basic File Structure

The basic structure of the file is that it contains general information about the session first in the *loghead* record, followed by blocks of task data or questionnaire data or individual events. Tasks can contain events or questionnaires. And finally, questionnaires contain responses. Notes from the subject or tester or others can be reported wherever events are legal. Here is a diagram of the constraints on the nesting structure:



2.4 Naming Convention

It is recommended that logfiles conforming to this specification be named with a file suffix of "ulog", e.g.

"session37.ulog".

3. Conformance

Several levels of conformance for logfiles are defined, involving both syntax and semantics. Although this specification contains some comments about the recommended properties of logfile generators and interpreters, we do not define conformance for such software.

Syntactic

A file conforms syntactically if it can be parsed according to the rules in the grammar section. The rules include the context-free grammar, context-sensitive rules, and explanantion of unresolved non-terminal symbols.

Defined Syntax

A file conforms to the defined syntax if it conforms syntactically and it contains no Ad_hoc elements. Thus, such a file can be parsed using only the defined grammatical fields. Generic interpreters should be able to completely analyze such a file.

Semantically true

A logfile is *semantically true* iff

1. it conforms syntactically
2. all the tasks, questionnaires, events, and notes represented in the file actually occurred as described
3. all the Ad_hoc elements represented in the file actually occurred as described, according to the semantic meaning implicit in the file generator
4. all the identifier elements in the file serve as identifiers: there is an accurate one-to-one mapping between these syntactic elements and the entities within the session that they purport to name.

The idea here is that everything the file is asserting about reality actually happened, even though there might be omissions ("The truth and nothing but the truth, but not necessarily the whole truth"). For defined elements, their meaning is established by this specification. For ad hoc elements, their meaning is established by the generator: it must define what it intends if it says, e.g. "glorp=2". Identifiers must be unambiguous, i.e. it is impermissible to have two names for the same entity, or the same name for two distinct entities.

Semantically state-complete

A file is semantically state-complete if it is semantically true and if all defined events that actually occurred in the session are recorded in the file, except those that involve some continuous operation, namely: pointer_moves, intermediate positions for scrolling and sliders, intermediate states of window moving and re-sizing, and six_DOF (degree of freedom) adjustments.

Semantically complete

A file is semantically complete if it is semantically true and if all defined events that actually occurred in the session are recorded in the file.

Note that checking semantic conformance is a fairly difficult task. But this difficulty follows from the fact that the logfile purports to be a description of the real-world behavior of a user of some software system. Semantic conformance necessarily involves some comparison between that description and the real-world

events that are its subject.

---

4. Specifications

The specification sections describe:

- Logfile
- Tasks
- Events
- Questionnaires
- Low-level constructs

Each logfile record is a sequence of fields, which are separated by spaces. Thus, any spaces within a field must be quoted, "like this". The general format for each record is:

```
Recordtype Mandatory-simple Mandatory-defined Optional-defined Ad-hoc
```

- The Recordtype declares what kind of a record this is, e.g. "loghead", "event", "taskend".
- Mandatory-simple fields come next and are unlabeled values.
- Mandatory-defined fields follow, and are labelled with defined keywords (e.g. widget=abc).
- Optional-defined fields follow, and are labelled with defined keywords but of course they need not be present.
- Ad-hoc fields come last - they are arbitrary keyword-value pairs.

The formal grammar presented below uses conventional symbols:

```
*  == 0 to infinity occurrences
+  == 1 to infinity occurrences
?  == 0 to 1 occurrences
|  == or
() == group syntactic elements
```

Variables (non-terminals) are capitalized, terminal symbols are lowercase or "Quoted". Comments (usually context-sensitive constraints) are [in brackets].

Specification: Logfile

Syntax

```
Logfile              == Log_header Log_block* Log_end
                        [all times within the file must be in weakly
                         ascending order.]

Log_header           == loghead Begin_time subject=Subject_ID
                        version=Log_version time_zone=Time_zone Generator_ID?
                        Browser_hardware? Browser_software?
                        Tester_ID? Procedure? Session_set_ID?
                        Detected_events? Ad_hoc_region

Log_end              == logend End_time Ad_hoc_region
```

```
Begin_time           == Time_value
End_time             == Time_value
Time_value           == Year/Month/Day-Hour:Minute:Seconds
Year                 == Integer [all 4 digits]
Month                == Integer [1 - 12]
Day                  == Integer [1 - 31]
Hour                 == Integer [0 - 23 ... e.g. 15 for 3pm]
Minute               == Integer [0 - 59]
Seconds              == Real    [0.0 - 59.9999]

Subject_ID           == Alpha
Log_version          == v1
Time_zone            == Number
Generator_ID         == generator=Alpha
Browser_hardware     == browser_hw=Alpha
Browser_software     == browser_sw=Alpha
Tester_ID            == tester=Alpha
Procedure            == procedure=Alpha
Session_set_ID       == sset=Alpha

Detected_events      == Defined_detection Ad_hoc_detection
Defined_detection    == detected_U=Def_user_list        ?
                        detected_W=Def_widget_list      ?
                        detected_OW=Def_widget_list     ?
                        detected_WN=Def_window_op_list  ?
                        detected_OP=Def_web_op_list     ?

Def_user_list        == defined_user_action(,defined_user_action)*
Def_widget_list      == defined_widget(,defined_widget)*
Def_window_op_list   == defined_window_op(,defined_window_op)*
Def_web_op_list      == defined_web_op(,defined_web_op)*

defined_user_action  == keypress | keyrelease | pointerpress |
                        pointerrelease | pointermove | six_DOF |
                        enterwidget | leavewidget |
                        enterwindow | leavewindow
defined_widget       == button | radio | checkbox | textbox | menu |
                        slider | scrollbarH | scrollbarV | handle | link
defined_window_op    == open | close | icon | de_icon | move | resize |
                        raise | lower | focus | blur
defined_web_op       == print | newpage:request | newpage:loading |
                        newpage:complete | page_locate

Ad_hoc_detection     == detected_undef_U=Label_list   ?
                        detected_undef_W=Label_list   ?
                        detected_undef_OW=Label_list  ?
                        detected_undef_WN=Label_list  ?
                        detected_undef_OP=Label_list  ?

Label_list           == Label(,Label)*

Ad_hoc_region        == Ad_hoc_fields Newline
Ad_hoc_fields        == (Keyword=Value)*
                        [Keyword must not match a defined keyword]
Keyword              == Label
Value                == Alpha
Number               == Signed_Integer | Signed_Real

Log_block            == Task_block | Questionnaire_block | Event_record |
```

```
                        Note_record | Comment_record

Note_record          == note Begin_time Author Note_content Ad_hoc_region
Author               == subject | tester | unknown | Alpha
Note_content         == Alpha

Comment_record       == "--" Any_character* Newline
                        [The 3rd character of the record (immediately
                         following the initial "--") need not be blank.
```

Semantics

Logfile

Describes a usability testing session. We define a *session* as the interaction of a single subject with a single fully configured hardware system during a continuous time interval. A switch of platform or subject is therefore considered to be a new session, by definition.

Log_header

Contains information global to this session.

Begin_time, End_time

The times at which a session, or some other entity with duration, begins and ends.

Time_value

Structured representation of a point in time. Seconds must be in fractional format even if the resolution is less than that. E.g. "45.0" rather than "45".

Subject_ID

A string that uniquely identifies the person whose activity is being logged. This could be a label pointing to a file or some other data entry with a full demographic description.

Log_version

Describes which version of this specification the file conforms to. For now, there's only one possibility, namely "v1".

Time_zone

Identify the time zone in which the time_values of the session are expressed. E.g. -5 for Eastern Standard Time. Subtracting the time zone (expressed in hours) from the time_values yields universal time. Real numbers are allowed because some zones are on the half-hour.

Generator_ID

Identify the logging software that wrote this file.

Browser_hardware

Information about the client hardware on which the session took place. Such things as machine speed and screen characteristics can be captured here. This may be useful for comparison among sessions.

Browser_software

Identifies such information as the browser version and operating system.

Tester_ID

This field identifies the person (normally a usability engineer) primarily responsible for setting up the test session.

Procedure

This field describes or points to the procedure used to set up and perform the test session.

Information such as how the subject was instructed, the general setting of the room, etc, may be entered here.

Session_set_ID

This allows several interacting sessions to be associated. E.g. if several subjects are testing collaborative software, their sessions could all be tagged with the same session_set_ID, to allow automated analysis of their interaction.

Detected_events

This parameter is used by the generator to declare that it has reported all event aspects of a certain type. Thus, if no instances of that type occur in the file, one may conclude that none occurred in the actual session.

Defined_detection

This is used to declare which of the standard-defined components and sub-components of events are guaranteed to be detected and reported. For instance,

```
detected_W=checkbox,radio,handle
```

asserts that all direct interactions with checkboxes, radio buttons, and handles (as defined below) will be detected and reported.

Ad_hoc_detection

This is used to declare that certain ad hoc (known to the generator) components and sub-components of events are guaranteed to be detected and reported.

Log_end

While there are no defined fields other than End_time, Log_end may also be used to capture summary data and information specific to the session, e.g.

```
number_tasks_completed=4
```

Ad_hoc_fields

Allows writers and readers of a logfile to convey information unanticipated by this specification. For instance, if the color of the room in which the session is held is deemed significant, the Log_header could legally contain: "room-color=yellow". Generic logfile readers/analyzers are expected to ignore these fields.

Note_record

This record captures information typed in during the session by the subject or tester or some other Author. The idea is that test session manager software might provide a facility for observations, comments, complaints, or recommendations by interested parties if they encounter some unusual situation. This could be invoked at the initiative of the note's Author or prompted by the system. The Begin_time is the time at which the author started composing the note. If the subject is the Author, therefore, he/she was not engaged in the application task.

Comment_record

This record has no defined meaning and generally should be skipped by software that uses the logfile as input. It may be inserted and/or read by a human operator to clarify the meaning of nearby logfile records or other purposes.

Specification: Tasks

Syntax

```
Task_block          == Task_header Task_entry* Task_summary
Task_header         == taskhead Begin_time Task_type_ID
                       website=Website_ID testdata=Testdata_ID
                       Ad_hoc_region
Task_type_ID        == Alpha
Website_ID          == Alpha
Testdata_ID         == Alpha

Task_entry          == Event_record | Questionnaire_block |
                       Note_record | Comment_record

Task_summary        == taskend End_time Task_type_ID Ad_hoc_region
                       [Task_type_ID must match Task_type_ID
                        from Task_header]
```

Semantics

Task_block

The FLUD file format is designed for task-oriented usability testing: the subject is given a task to perform (e.g. find at least three documents about Iowa, find out how much a Boeing 747 weighs) and then his/her performance is monitored. Undirected browsing can also be recorded within a single "dummy" task.

Task_header

Holds information pertaining to an entire single task within the session.

Task_type_ID

Uniquely identifies the type of task that the subject is to perform. For instance, the task of discovering the weight of a 747 might be labeled as "find-weight-747". There might well be several *instances* of the same task type within a session if the subject is asked to perform the same task under varying circumstances. Also, this allows comparison of task performance across sessions.

Website_ID

Not necessarily an URL, although it may be - this identifies the whole site or sub-site being tested, not just a single page. Successful performance of the task may require the subject to visit several pages. Also, this might be used to identify which *version* of a website is being tested.

Testdata_ID

Identifies the underlying data file(s) or database with which the subject is interacting in this task. It may be the name of a file or directory.

Task_summary

Denotes completion of task (whether successfully accomplished or not). An application-savvy generator may well want to use ad hoc fields here to report summary metrics from the task, e.g. number_found=6.

Specification: Events

Syntax

```
Event_record          == event Event_header User_widget_effect Newline
Event_header          == Event_time Window_ID URL?
Event_time            == Time_value | Partial_time_value
Partial_time_value    == Hour:Minute:Seconds
                         [inherits the most recently specified date -
                          careful: need full Time_value for midnight rollover]
Window_ID             == Alpha
URL                   == url=Alpha

User_widget_effect    == User_action? This_widget? System_effect*
                         [at least one of these three must be present.]

User_action           == #U (Defined_user_action | Ad_hoc_user_action)
                         Ad_hoc_fields

Defined_user_action   == Key_press_event     | Key_release_event |
                         Pointer_press_event | Pointer_release_event |
                         Pointer_move_event  | Six_DOF_event |
                         Enter_widget_event  | Leave_widget_event |
                         Enter_window_event  | Leave_window_event

Key_press_event       == keypress   XY_location Key_ID
Key_release_event     == keyrelease XY_location Key_ID
Key_ID                == key=Key_data
Key_data              == Key_mod* (Key_character | Key_label)
Key_mod               == shift- | ctl- | meta- | alt-
Key_label             == f1 | f2 | ... | sp | vt | tab | ff | cr | ht |
                         lf | bs | esc | page_up | page_down |
                         arrow_up | arrow_down | arrow_left | arrow_right |
                         home | end | insert | delete | Plain_char Plain_char+
                         [Key_label must be at least 2 characters to
                          distinguish it from Key_character.  The space
                          character must be represented as the label "sp",
                          not as an actual space.]

Pointer_press_event   == pointerpress   XY_location Button_ID
Pointer_release_event== pointerrelease XY_location Button_ID
Button_ID             == button=Button_data
Button_data           == Key_mod* Integer
                         [start at 1: 3-button mouse is labelled as 1,2,3,
                          not 0,1,2]
Pointer_move_event    == pointermove  XY_location

Six_DOF_event         == six_DOF XY_location?
                         shift=Three_D_data rotate=Three_D_data
Three_D_data          == Signed_Real,Signed_Real,Signed_Real

Enter_widget_event    == enterwidget
Leave_widget_event    == leavewidget
                         [Event_record must contain This_widget field if
                          enter or leave widget is reported.]

Enter_window_event    == enterwindow
Leave_window_event    == leavewindow

Ad_hoc_user_action    == User_action_type XY_location?
User_action_type      == Label

XY_location           == screen_xy=XY_value | window_xy=XY_value |
                         webpage_xy=XY_value
```

```
This_widget          == #W Widget_info Ad_hoc_fields
Widget_info          == Widget_ID Widget_type_value Widget_level?
                        Widget_ready?

Widget_ID            == Alpha
Widget_type_value    == button Bvalue? | radio Bvalue? | checkbox Bvalue? |
                        textbox TXvalue? | menu Avalue? | slider Nvalue? |
                        scrollbarH SCvalue? | scrollbarV SCvalue? |
                        handle Bvalue? | link Bvalue? | Label Avalue?
Bvalue               == value=Boolean
Avalue               == value=Alpha
Nvalue               == value=Number
SCvalue              == value=Scroll_value
TXvalue              == value=Text_value

Scroll_value         == Real,Real
Text_value           == Quoted_string(:Cursor_position)? | :Cursor_position
Cursor_position      == XY_value

Widget_level         == level=(page | browser | window | system | Label)
Widget_ready         == ready=(focus | blur | hidden | displayed | Label)

System_effect        == (Other_widget | Window_state | Webpage_operation)
                        Ad_hoc_fields

Other_widget         == #OW Widget_info

Window_state         == #WN Window_spec Window_op
Window_spec          == "*" | Window_ID
Window_op            == open Rectangle_position? | close |
                        icon | de_icon Rectangle_position? |
                        move Rectangle_position | resize Rectangle_position |
                        raise | lower | focus | blur | Label
Rectangle_position   == up_left=XY_value low_right=XY_value

Webpage_operation    == #OP (Defined_web_op | Ad_hoc_web_op)
Ad_hoc_web_op        == Label
Defined_web_op       == print URL? |
                        newpage Newpage_op Window_spec URL |
                        page_locate Window_spec URL Page_loc_info
Page_loc_info        == horizontal=Scroll_value (vertical=Scroll_value)? |
                        vertical=Scroll_value

Newpage_op           == request | loading | complete | Label
```

Summary of defined events

- User_action
  - Key_press_event, Key_release_event
  - Pointer_press_event, Pointer_release_event
  - Pointer_move_event
  - Six_DOF_event
  - Enter_widget_event, Leave_widget_event
  - Enter_window_event, Leave_window_event

- This_widget
  - button
  - radio
  - checkbox
  - textbox
  - menu
  - slider
  - scrollbarH, scrollbarV
  - handle
  - link
- System_effect
  - Other_widget [entries same as This_widget]
  - Window_state
    - open, close
    - icon, de_icon
    - move, resize
    - raise, lower
    - focus, blur
  - Webpage_operation
    - print
    - newpage
      - request
      - loading
      - complete
    - page_locate

Semantics

Event_record

Describes a single event. An event is considered to be nearly instantaneous and so only one time applies to it. More complex actions with a significant duration (such as drag-and-drop) are represented as a sequence of events. There are three kinds of substantive components of an event: User_action, This_widget and System_effects (see below). These are related causally; they are not merely co-temporal. An example would be a user clicking on a "close window" button. The user_action is the mouse click, the widget is the labelled button, and the effect is the actual closing of the window.

An event_record need not contain all three components. A component may be omitted because there is no information to report (e.g. no This_widget field because the user_action is not directed at a widget) or because the generator chooses not to report it. Moreover, a causally related user_action and system_effect may be reported within a single Event_record or in several. E.g. if the user clicks on a link, the resulting page may not show up for some time thereafter, and so its appearance may be reported in a later record as a separate event.

Event_header

These fields apply by default to all the components of the event_record. The window and URL information may be overridden by more specific information within a component.

Event_time

Events are considered to be instantaneous (zero duration), and therefore have a single time-stamp, not a begin and end time.

Partial_time_value

Inherits date information from most recent explicit Time_value in the preceding records of the file. Midnight rollover must be reported explicitly, i.e. "11:59:50.1" should be followed by something like "2000/05/17-00:00:22.3", not just "00:00:22.3". All time values within the file must be greater than or equal to the preceding value.

Window_ID

The Window_ID field in the event_header indicates where the user action is directed, where the widget is located, and (usually) where the effects take place. If a different window is affected, this is denoted in a field within System_effects. When representing user interaction with a webpage with multiple frames, it is recommmended that the frame be identified as a subwindow through the use of an appropriate name, e.g. "win_237.frame_02".

URL

The URL field likewise indicates which URL is the target of the user action and (by default) where the effects take place. The value of this field is a fully-qualified URL. If a webpage contains relative URLs, it is the responsibility of the generator to resolve these into absolute URLs before recording them in the logfile. The URL field is optional for two reasons: first, the user_action may not be directed to an URL, but rather to some other software component, such as a browser button. Second, the logfile has an implicit window-to-URL mapping at any point, based on the newpage system effects, and so one can infer the URL, given the window.

User_action

A user_action is performed directly by the user and is associated with a particular input device, typically a mouse or a keyboard.

Key_press_event, Key_release_event

User presses or releases a key on a keyboard. XY_location is determined by the system pointer. Key_ID indicates which key was pressed/released. Key_character is used to represent "normal" printable single characters, such as 'W' or '#'. Key_label is used as the name of an unprintable character. Well-known names include "ff" for form-feed (ASCII 12), "bs" for backspace (ASCII 8) and so on.

Pointer_press_event, Pointer_release_event, Pointer_move_event

User presses or releases a button on system pointer (e.g. mouse), or simply moves it.

Six_DOF_event

User manipulates a device with six degrees of freedom (such as a spaceball). Such devices are often used in 3D applications to model rigid transformations (shift and rotate). XY_location is determined by the system pointer.

Enter_widget_event, Leave_widget_event

User moves system pointer into or out of a widget. The event_record must contain a this_widget component to identify which widget was affected.

Enter_window_event, Leave_window_event

User moves system pointer into or out of a window. The affected window is specified by the Window_ID in the Event_record header.

XY_location

Location in pixels, where user_action is directed. Location may be relative to the entire screen, the window of the event_header, or the webpage of the event_header.

This_widget

Describes the widget, if any, to which the user_action was targeted. Screen objects such as buttons, textboxes, menus, and sliders are typical widgets. Note that while a user_action is always directed towards at most a single widget, several widgets could be affected as a result. E.g. clicking a single "Clear" button may cause various textboxes, radiobuttons, and checkboxes to change state. Multiple effects may be represented in the Other_widget field of system_effects.

Widget_ID

Uniquely identifies a widget. Two distinct window areas with the same function are considered to be distinct widgets. E.g. if there is a "return to index" button at both the top and bottom of a webpage, these are distinct widgets with distinct names. It is recommended that:

- The widget name map in a reasonable way to a visible label, when one exists. E.g. a "reload" button could be named "win_01.reload".
- Names of radiobutton widgets reflect the related set to which they belong, e.g. "pizza_size.small", "pizza_size.medium", "pizza_size.large".
- Names of checkbox widgets reflect the related set to which they belong, e.g. "pizza_topping.anchovy", "pizza_topping.pepperoni", "pizza_topping.mushroom".

Widget_type_value

The widget type determines what values and operations are available for that widget:

button

Buttons have no state - they simply initiate some action: e.g. do, calculate, search, submit, clear. A special case is a button that opens a menu. Buttons take a Boolean value, where "yes" indicates the button was pressed, and "no" that it was not.

radio

Radiobuttons have a binary state and are used to indicate a "one of many" selection, e.g. pizza size: small, medium or large. Each radiobutton toggles on and off; toggling on causes other radiobuttons in the related set to turn off. It takes a Boolean value, where "yes" indicates the toggle on and "no" is toggle off.

checkbox

Checkboxes have a binary state and are used to indicate a many of many selection, e.g. pizza toppings: anchovies, pepperoni, mushroom. Each checkbox toggles on and off; toggling has no effect on other checkboxes in the related set. It takes a Boolean value, where "yes" indicates the toggle on and "no" is toggle off.

textbox

Textboxes accepts a textual entry, usually with editing allowed. The contents may consist of several lines. The state is the current text contents and location of text cursor. Textbox takes a Text_value value indicating current contents and/or text cursor position. The units of the Cursor_position field are interpreted as characters and lines, rather than pixels. If the cursor is in front of the first character of the first line, its position is 0,0. If in front of the 3rd character on the 7th line, its position is 2,6.

menu

A menu allows selection of one of many displayed choices, but the state not retained. It takes an Alpha value, indicating which selection was made. Although functionally

somewhat similar to a set of radiobuttons, a menu is treated as a single widget.

slider

Lets the user adjust some continuous one-dimensional numeric quantity (integer or real). Its state is the current numeric value. It takes a Number value.

scrollbarH, scrollbarV

This widget is a slider on the border of a window, used to adjust the position of a page within the window: scrollbarH for horizontal or scrollbarV for vertical. Its state value is the sub-page implicitly requested by the widget. Two numbers are needed to characterize Scroll_value, meaning: fraction of webpage cropped at top/left, and fraction cropped at bottom/right. E.g. if lines 23 through 65 of a 100-line document are requested, the vertical crop values would be 0.22 (top) and 0.35 (bottom).

handle

A handle is used for dragging to adjust window geometry, e.g. for moving or re-sizing a window. There is no state. A handle takes a Boolean value, where "yes" indicates the handle is being actively used and "no" that it is not.

link

A link is a region (text or image) of a webpage made active by the presence of an anchor element (i.e. an HREF). Clicking on a link typically requests the retrieval of some new web resource or some other action. Examples include the display of a new webpage or new location within the current webpage, launching a CGI script, sending mail, and so on. These resulting actions, however, are represented in the System_effect field. The link *per se* is simply the clickable region that triggers the action. Link takes a Boolean value, where "yes" indicates the link was activated and "no" that it was not. The Widget_ID identifies the region of the page used as a link.

Widget_level

The component directly containing the widget at which the user action is targeted.

page

The widget is part of a displayed webpage, e.g. a checkbox.

browser

The widget is one of the standard browser controls, e.g. the back button or a scroll bar to adjust page position.

window

The widget is one of the standard window controls, e.g. the resize handle.

system

The widget is one of the operating system controls, e.g. a button to launch a new process.

Widget_ready

Describes the visibility and/or availability of the widget to accept user input:

focus

The widget is ready to accept user input, e.g. an active textbox that gets input from the keyboard. Focus implies displayed.

blur

The widget is not ready to accept user input, e.g. an inactive textbox.

hidden

The widget has become hidden, e.g. when a pull-down menu is collapsed.

displayed

The widget has become visible, e.g. when a user_action causes a menu to be displayed.

System_effect

System_effect is used to describe how the state of the system as seen by the user changes (either as a result of the user_action or autonomously) other than immediate changes in the state of the widget directly targeted.

Other_widget

The components of other_widget have the same interpretation as in This_widget, except that the other_widget changed state as an *indirect* effect of a user_action. E.g. if the user presses a "Clear" button, that button is directly affected, but other widgets may change state as a result - textboxes cleared to the null string, checkboxes set to "off", etc. These other changes could be packaged up into one event_record along with the targeted widget, they could be represented in a distinct event_record, or they could each have their own event_record. In the latter two cases, however, the causal connection between the original user_action and the resultant widget state changes would be more difficult to reconstruct.

Window_state

Reports changes in the state of one of the windows of the web application.

Window_spec

A window_spec of "*" means that the Window_ID in the event_header is the one in which this state change has occurred. An explicit Window_ID in the Window_spec overrides the Window_ID in the event_header.

Window_op

The defined window operations are as follows. When rectangle_position is given, it indicates the location of the window in pixel coordinates relative to the full screen.

open

A new window appears on the screen.

close

An existing window is deleted from the screen.

icon

The window is iconified - represented as a small region and temporarily de-activated.

de_icon

The window is de-iconified - restored to full size from an iconified state.

move

The window's position on the screen changes.

resize

The window's size and possibly shape on the screen changes.

raise

The window is raised so as to be fully visible and therefore may obscure other windows.

lower

The window is lowered so as to be potentially obscured (partially or fully) by other windows.

focus

The window is ready to accept user input, e.g. from the keyboard.

blur

The window will not accept certain user input, e.g. from the keyboard.

Webpage_operation

Webpage operations include loading of webpages but also any application-specific operations that can reasonably be detected and represented. A description of the defined webpage operations (defined_web_op) follows:

print

A webpage is printed out. The URL to be printed out is taken from (in order of preference), an explicit URL field, the URL of the event_header, or the URL associated with the window in the event_header.

newpage

Newpage encompasses events surrounding the retrieval and display of a new web page. The URL is that of the new page. Note that jumping to an *internal* link normally results in a page_locate effect (see below), not a newpage. The window_spec indicates in which window the URL will be displayed. The defined newpage operations include:

request

The user has issued a request for a webpage. The webpage may or may not exist and may or may not be successfully retrieved.

loading

The webpage has been found and its retrieval and display have begun.

complete

The webpage has been found and its retrieval and display have been completed.

page_locate

A webpage (specified by the URL field) changes location within a window (specified by Window_spec), perhaps as a result of page scrolling, window re-sizing, or jumping to an internal link. The horizontal and vertical parameters describe the precentage of the page that is cropped at the left/top and right/bottom.

Specification: Questionnaires

Syntax

```
Questionnaire_block  == Q_header Q_entry* Q_end
Q_header             == qhead Begin_time Questionnaire_ID Ad_hoc_region
Questionnaire_ID     == Alpha
Q_entry              == Q_record | Comment_record
Q_record             == qrec Question_ID Response Ad_hoc_region
Question_ID          == Alpha
Response             == Alpha
Q_end                == qend End_time Ad_hoc_region
```

Semantics

Questionnaire_block

This captures the results of a portion of the session wherein the subject responds to a questionnaire set up by the tester. The difference between a questionnaire and a task is that a questionnaire requests information directly from the subject (e.g. "how old are you?", "do you think the graphics are helpful or annoying?"), whereas a task is usually meant to simulate the intended usage of the website and the subject is monitored to find out such things as whether

most people use the website effectively. Also, only the results of the questionnaire are reported, not the process by which they were answered (e.g. timing of the responses is not reported).

Q_header

Report with which questionnaire the subject is interacting, and when the interaction started.

Q_record

Each Q_record captures the response to a single question of the questionnaire. The intention is that the Question_ID simply identifies which question is being answered, rather than repeating the entire question verbatim. All questions answered during the time span of the questionnaire must be reported. Only the final responses of the user should be reported (e.g. if the user first clicks "green", then "red" in response to a question, only the latter should be reported). If the user does not give a response to a question, this may be reported either by simply omitting the Q_record for that question, or by a Response of "" (an empty Quoted_string).

Q_end

Report when the user's interaction with the questionnaire ended.

Specification: Low-level constructs

Syntax

```
Alpha               == Label | Quoted_string
Label               == Plain_char+
Plain_char          == Letter | Digit | - | _ | . | :
Quoted_string       == Double_quote (DQ_character | print_item |
                       unprint_item)* Double_quote
Double_quote        == "
print_item          == \" | \\
unprint_item        == \ Letter
XY_value            == Signed_Integer,Signed_Integer
Signed_Real         == Optional_sign Real
Signed_Integer      == Optional_sign Integer
                       [no spaces within signed numbers]
Real                == Integer . Integer?
Integer             == Digit+
Optional_sign       == ("+" | "-")?
Boolean             == yes | no
Letter              == A | B | C | ... | Z | a | b | c | ... | z |
Digit               == 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Semantics

Label (%s)

An unquoted alphabetic string. May be used as a self-explanatory literal or as a pointer to some fuller description. E.g. a Subject_ID with a value of *JoeJones* could be treated as a simple designator of a person or used as a filename, with the file containing a fuller description of the person (birth date, sex, SSN, etc.).

Quoted_string

A representation of a string of characters enclosed in double quotes ("). An Alpha entry must be quoted if it contains non-plain characters. DQ_characters represent themselves. Two printable characters cannot be self-represented, the double-quote and the backslash, so these are preceded

by a backslash. Non-printable characters within a string, such as newline or backspace, are represented using the backslash conventions of the C standard. The meaning of any backslash sequence other than those in the table below is defined by the generator.

backslash sequence	meaning	printable character
\"	double-quote	yes
\\	backslash	yes
\a	alert	no
\b	backspace	no
\f	form feed	no
\n	newline	no
\r	carriage return	no
\t	tab	no
\v	vertical tab	no

For example, if the user typed these two lines in a textbox widget:

```
abc"def
line\2
```

The value for the textbox would be represented in a FLUD file as:

```
"abc\"def\nline\\2"
```

A quoted_string could be used as a pointer or filename, but also as a literal description.

XY_value

An ordered pair of integers used to locate an interaction within a window. The first number (X) specifies a horizontal location, and the second (Y), a vertical location.

Real (%f)

A real number, fixed-point format (must have decimal point).

Integer (%d)

A non-negative integer.

Context-sensitive grammar rules.

- All Time_values and Partial_time_values within the file must be in ascending order, i.e. If time X textually precedes time Y within the file, then the time designated by X must be no later than that designated by Y.
- The Year field is interpreted as a complete value - no implicit centuries. E.g. the year 1998 is represented by "1998" or even "001998", not simply "98".
- All Time_values and Partial_time_values must designate actual times: Month between 1 and 12, Day between 1 and number of days in the month, Hour between 0 and 23, Minute between 0 and 59, 0 <= Seconds < 60.
- Keywords in an Ad_hoc_field must not match any defined keywords.
- The Task_type_IDs of a Task_header and Task_summary within the same Task_block must be the same.

Explanation of unresolved non-terminal symbols

DQ_character

A DQ_character is any printable character, including space, other than a Double_quote (") or backslash (\).

Any_character (%c)

Any character other than newline.

Key_character (%c)

Any non-whitespace (printable) keyboard character.

Newline

A newline character, signifying end of a record.

5. Example

Click here to see an example of a FLUD file.