PROVISIONAL APPLICATION FOR PATENT COVER SHEET

This is a request for filing a PROVISIONAL APPLICATION FOR PATENT under 37 CFR 1.53(c).

INVENTOR(S)

Given NameFamily Name or Surname   Residence

Sumeet Sohan Singh   Saratoga, California

TITLE OF THE INVENTION (500 characters max):

A method of manually filling forms and training a form filler application using Drag and Drop UI gestures

CORRESPONDENCE ADDRESS

Direct all correspondence to:

_X_ The address corresponding to Customer Number: _115980______

OR

__Firm or Individual Name: Sumeet Sohan Singh

Address: 18518 Montpere Way
City: Saratoga
State: California
Zip: 95070
Country: United States (US)
Telephone: 650-520-8656
Email: sumeet@singhonline.info

ENCLOSED APPLICATION PARTS (check all that apply)

___ Application Data Sheet. See 37 CFR 1.76___ CD(s), Number of CDs ______________

_X__ Drawing(s) Number of Sheets _6______ ___   Other (specify): ______________

_X_   Specification (e.g. description of the invention) Number of Pages _9______

METHOD OF PAYMENT OF THE FILING FEE AND APPLICATION SIZE FEE FOR THIS PROVISIONAL APPLICATION FOR PATENT

X Applicant claims small entity status. See 37 CFR 1.27. **TOTAL FEE AMOUNT ($)**

___ A check or money order made payable to the *Director of the United States Patent and Trademark Office* is enclosed to cover the filing fee and application size fee (if applicable) **$125**

___ Payment by credit card. Form PTO-2038 is attached.

___ The Director is hereby authorized to charge the filing fee and application size fee (if applicable) or credit any overpayment to Deposit Account Number: __________________________________.

USE ONLY FOR FILING A PROVISIONAL APPLICATION FOR PATENT

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

PROVISIONAL APPLICATION COVER SHEET

The invention was made by an agency of the United States Government or under a contract with an agency of the United States Government.

X No.

___ Yes, the name of the U.S. Government agency and the Government contract number are: ________________________

WARNING:

Petitioner/applicant is cautioned to avoid submitting personal information in documents filed in a patent application that may contribute to identity theft. Personal information such as social security numbers, bank account numbers, or credit card numbers (other than a check or credit card authorization form PTO-2038 submitted for payment purposes) is never required by the USPTO to support a petition or an application. If this type of personal information is included in documents submitted to the USPTO, petitioners/applicants should consider redacting such personal information from the documents before submitting them to the USPTO. Petitioner/applicant is advised that the record of a patent application is available to the public after publication of the application (unless a non-publication request in compliance with 37 CFR 1.213(a) is made in the application) or issuance of a patent. Furthermore, the record from an abandoned application may also be available to the public if the application is referenced in a published application or an issued patent (see 37 CFR 1.14). Checks and credit card authorization forms PTO-2038 submitted for payment purposes are not retained in the application file and therefore are not publicly available.

SIGNATURE [illegible] Date June/17/2013

TYPED or PRINTED NAME Sumeet Sohan Singh REGISTRATION NO. _________
(if applicable)

TELEPHONE 650-520-8656 Docket Number: DND-1

Privacy Act Statement

The **Privacy Act of 1974 (P.L. 93-579)** requires that you be given certain information in connection with your submission of the attached form related to a patent application or patent. Accordingly, pursuant to the requirements of the Act, please be advised that: (1) the general authority for the collection of this information is 35 U.S.C. 2(b)(2); (2) furnishing of the information solicited is voluntary; and (3) the principal purpose for which the information is used by the U.S. Patent and Trademark Office is to process and/or examine your submission related to a patent application or patent. If you do not furnish the requested information, the U.S. Patent and Trademark Office may not be able to process and/or examine your submission, which may result in termination of proceedings or abandonment of the application or expiration of the patent.

The information provided by you in this form will be subject to the following routine uses:

1. The information on this form will be treated confidentially to the extent allowed under the Freedom of Information Act (5 U.S.C. 552) and the Privacy Act (5 U.S.C 552a). Records from this system of records may be disclosed to the Department of Justice to determine whether disclosure of these records is required by the Freedom of Information Act.
2. A record from this system of records may be disclosed, as a routine use, in the course of presenting evidence to a court, magistrate, or administrative tribunal, including disclosures to opposing counsel in the course of settlement negotiations.
3. A record in this system of records may be disclosed, as a routine use, to a Member of Congress submitting a request involving an individual, to whom the record pertains, when the individual has requested assistance from the Member with respect to the subject matter of the record.
4. A record in this system of records may be disclosed, as a routine use, to a contractor of the Agency having need for the information in order to perform a contract. Recipients of information shall be required to comply with the requirements of the Privacy Act of 1974, as amended, pursuant to 5 U.S.C. 552a(m).
5. A record related to an International Application filed under the Patent Cooperation Treaty in this system of records may be disclosed, as a routine use, to the International Bureau of the World Intellectual Property Organization, pursuant to the Patent Cooperation Treaty.
6. A record in this system of records may be disclosed, as a routine use, to another federal agency for purposes of National Security review (35 U.S.C. 181) and for review pursuant to the Atomic Energy Act (42 U.S.C. 218(c)).
7. A record from this system of records may be disclosed, as a routine use, to the Administrator, General Services, or his/her designee, during an inspection of records conducted by GSA as part of that agency's responsibility to recommend improvements in records management practices and programs, under authority of 44 U.S.C. 2904 and 2906. Such disclosure shall be made in accordance with the GSA regulations governing inspection of records for this purpose, and any other relevant (i.e., GSA or Commerce) directive. Such disclosure shall not be used to make determinations about individuals.
8. A record from this system of records may be disclosed, as a routine use, to the public after either publication of the application pursuant to 35 U.S.C. 122(b) or issuance of a patent pursuant to 35 U.S.C. 151. Further, a record may be disclosed, subject to the limitations of 37 CFR 1.14, as a routine use, to the public if the record was filed in an application which became abandoned or in which the proceedings were terminated and which application is referenced by either a published application, an application open to public inspection or an issued patent.
9. A record from this system of records may be disclosed, as a routine use, to a Federal, State, or local law enforcement agency, if the USPTO becomes aware of a violation or potential violation of law or regulation.

PROVISIONAL PATENT APPLICATION

Invention Type

This invention is an invention of the following type(s):
* Software

Invention Title

The title of the invention is: A method of manually filling forms and training a form filler application using Drag and Drop UI gestures.

Background

Several software applications are available that insert user input data into webpages and other applications and/or capture user input data from webpages and other applications. Webpages are usually a variant of HTML or XHTML markup languages. Software applications that perform such actions are popularly called 'password managers' or 'form fillers'. In this document they shall be referred as 'form-fillers' for brevity only but the term 'form-fillers' in this document shall encompass 'password managers' as well as any software program that performs the recognition, auto-fill and auto-capture functions described in this document.

Note: In this document, several times separates words are hyphenated for better readability. For example 'input field' may be written as 'input-field', 'input data' may be written as 'input-data', 'web page' as 'web-page' etc. The hyphenated combination of words should not be construed as a separate term requiring a separate definition. For example the term 'input field' is defined later in this document; therefore 'input-field' shall mean 'input field' as well.

Definition: The term 'application' in this document shall mean a 'software application' or a piece of 'software' that is identified as being separate from other software for some purpose. Examples are:

a) A web-application's software running inside a web-browser. A web-browser add-on, extension or plugin.
b) A standalone desktop, tablet or mobile application such as an email client.

Definition: The software eco-system in which a software application operates shall be called 'platform' or 'operating platform' in this document. Examples of 'platform' are:

1. The operating system inside which a software application operates, e.g. Microsoft Windows, Apple OSX and Linux,
2. The web-browser or web-page inside which a web-page script, web-browser extension, add-on or plugin operates. This definition includes web-browsers running on any computing device including desktops, laptops, netbooks, tablets, mobile phones and PDAs. Examples of web-browsers are Internet Explorer, Google Chrome, Mozilla Firefox, Apple Safari and Opera.

Definition: Software application programming interface extended by the platform shall be called 'platform API'.

Definition: The term 'UI' or 'User Interface' in this document shall denote the user-interface or parts of the user-interface presented by software applications. The definition includes the visual elements of the user-interface such as input fields, buttons, drop-down menus, text etc. The definition also includes other types of user-interface elements such as audible elements – e.g. beeps and chimes - and tactile – e.g. touch and vibration interfaces.

Popular form-fillers and password-managers remember and store input values on behalf of the user on persistent storage such as a computer's hard-drive or internet-storage that's under their control and fill-in the input-fields on web-pages and UI of other software applications (such as standalone desktop or mobile applications). They are also inbuilt features of popular web browsers such as Internet Explorer, Google Chrome, Firefox and Safari.

These applications typically work in the following way:

<u>Capturing User Input</u>

1. <u>Watch relevant UI form controls</u> (also called user input fields, input fields or form control fields in this document) on a webpage or other applications which accept user input. Examples of such form controls are:
    a. text input fields such as
        i. Credential input fields: username, user-id, login-id, account number, membership number, password, PIN etc.
        ii. Address fields: street address, city, state, zip-code, telephone, email etc.
    b. Drop-down selectors, checkboxes, radio buttons, range slider. Example usages are:
        i. Select credit card type (Amex, Visa, MasterCard) in a billing form
        ii. Select country field in an address form
        iii. Select shoe size when ordering custom made shoes
        iv. Color picker
        v. choose pizza toppings when ordering customized pizza

    <u>Definition</u>: In this document the terms 'input field', 'user input field', 'form control', 'form field' and 'form control field' 'control' shall all be synonyms for UI form controls as described in the preceding paragraph.

    <u>Definition</u>: The user input that is provided by the user to a form control shall be called 'user input', 'user input data', 'input data' or 'input value' in this document. This could be textual input entered into a text-field such as a username or password field, a selection performed from a set of choices such as drop-down menus and radio buttons, an option selected such from controls like check-boxes or color-pickers or anything else that a form-control may expose as its value or state. These are also known as control-values or control-states and are all ultimately submitted to a software application (e.g. a web-application) for performing some action (such as logging the user into a web-site).

    <u>Definition</u>: Form control fields are typically grouped together for a particular purpose such as login-information submission, sign-up information submission, online order submission etc. Groups of such form control fields shall be called 'forms' in this document. For example, a login form will typically have a username/login-id and password text-input fields, optionally a 'remember me on this computer' check box control and a 'submit' button.

    <u>Form and Input-Field Recognition</u>: Form-Fillers are usually discerning about which form controls / input fields to watch. Some are smarter than the others and – using intelligent algorithms – are able to precisely pinpoint the forms and/or input fields that are worth watching and/or weed out those that are not useful for the specific purposes of the form-filler. Such 'smart' form-fillers may not capture unnecessary user input (e.g. a password manager app will not capture address information because it recognized the form as being irrelevant to the purpose of password management).

    <u>Definition</u>: For the purposes of this document, a form-filler's ability to determine the purpose of a given form or input-field shall be called 'categorization'.

    <u>Definition</u>: For the purposes of this document, a form-filler's ability to detect, identify and categorize a form, an input field, input-data or web-page shall be called 'recognition'. These capabilities may also be referenced individually and called 'form recognition', 'input-field

recognition', 'input recognition' and 'web-site recognition' respectively. Web-site recognition refers to the form-filler's ability to recognize a web-page as belonging to a particular web-site (e.g. www.facebook.com and apps.facebook.com will both be recognized as belonging to facebook.com).

Some other form-fillers are not that smart and simply cast a wide net catching a lot of unnecessary user-input in the process. They then ask the user (via user-interface prompts and such) whether or not the input should be stored and if so, under what field-category. In other words they ask the user to recognize the input-data, fields and forms. Needless to say this can get annoying for the end-user. Some such applications may claim to be generic form-fillers simply because they do not have the technology to automatically categorize forms and input-fields accurately enough.

2. Capture input values as they are input (or filled in) to the input-fields. These values may be filled-in (input) by an end-user or another form-filler software or even the same one. If the values are same as before, then they're usually discarded for sake of efficiency. Form-fillers employ a variety of techniques to capture input data. Exactly how that is done is not within the scope of this document. A point to note is that the input-data will also be categorized based on its purpose – with the same categories as the input-fields into which it is input.

3. Store into persistent storage, the user input data along with identifying information of the webpage. In addition to captured input-data, many form-fillers store identifying and - in some cases – recognizing/categorization information about the forms and/or input-fields. Identifying information of a web-page is usually its full URL or a part of it. Identifying information of a standalone desktop application may be queried from the operating system.

   Field Identifying Information
   Identifying information of an input-field or form on a web-page can be one or more of its HTML-DOM element characteristics (such as its DOM tag name) and DOM attributes and properties (such as 'id' and 'name' attributes of the DOM element). The DOM element may also be identified in relationship to its ancestor elements (such as the associated form element), sibling or child elements. The same type of information that is used in 'CSS selector patterns' to identify elements maybe used herein as well, although the desire here is to uniquely identify one field whereas CSS selectors are typically used to identify multiple elements. Identifying information of UI elements of non-HTML applications are determined per the specific platform APIs – a full survey of which is outside the scope of this document.

4. Field recognition is not a 100% accurate. Since web-pages are generated by a variety of different software frameworks, are tailored to different web-browsers and versions thereof, have text in a variety of human languages and use a variety of flavors of HTML, they do not adhere to any standard method of naming, identifying and categorizing input-fields (as defined in this document). Same is true for standalone applications written for varying operating systems and devices. Therefore, detecting, identifying and categorizing (i.e. recognizing) web-page forms and input-fields is an art, not science. Algorithms used by form-fillers may employ heuristic and artificial-intelligence algorithms and logic to recognize input-fields and forms. They may employ similar techniques to recognize different pages of a web-site. However, as with any AI algorithm that works with real world data, their accuracy is not 100% correct.

   This document introduces a novel way to work around the recognition failures of form-fillers by enabling the form-filler to collect user-feedback and improve its recognizing abilities.

   Definition: The ability of a form-filler to capture user-input from input-fields shall be called 'auto-capture' in this document.

Fill In Form Control Fields

1. Check in persistent storage if there is any input-data available for the input-fields on the webpage or application that the user is currently focusing on. The webpage, application and input-fields are identified using identifying information already described. Input data that was previously associated with these input-fields is now filled into the same input-fields.

2. As described earlier, a smarter form-filler may not only identify but also categorize the input-fields on any web page and standalone applications, even those that it has no prior knowledge of. Therefore, such form-fillers may never even store input-field recognizing information, because they don't need to. Example categories of input-fields are 'username' and 'password' categories. Correspondingly, input-data is also classified into the same categories – e.g. 'username' and 'password' categories. Such a 'smart' form-filler may be able to scan a given web-page or standalone application UI, identify and categorize input-fields there in, retrieve from its persistent data-store input-data pertinent to the web-page/application and input-fields, and fill-in the input-fields with or without express user-command. Alternatively, the form-filler may make the input-data available to the end-user in an easily consumable user-interface such as a visual array of usernames and passwords. A smart form-filler would be able to do all this without having any prior knowledge – i.e. without any prior recognizing information - of the web-page, the form or the input-fields.

   Please see Figure 1 for an example of a smart form-filler. It displays pertinent user-input on any web-page. It also allows auto form filling upon the click of a button (the black arrows pointing left). It also allows the user to drag-and-drop input-data into input-fields either on the web-page, on any other web-page, on a page in another browser, or on any standalone application with input-fields.

   In general, note that the input-data may either have been imported from another database, manually input by the end-user or captured by the form-filler as described in paragraph Capturing User Input. There is a lot of variation amongst the existing applications vis-à-vis their ability to recognize input-fields, recognize user-input data and recognize web-pages. However all that is not part of this invention and therefore this document will not go into those details.

3. In general, form-fillers use appropriate platform APIs - e.g. HTML DOM APIs when operating inside a web-browser page - to insert the data into input-fields. User interaction and/or permission is sought from the user as appropriate. When filling fields inside web-pages, the form-fillers usually embed their software into web-browsers in the form of browser add-ons / extensions and plugins. Such 'embedded software' can then access the web-page DOM APIs which are used to perform the Auto-Fill and Auto-Capture functions.

4. Form Filling is not a 100% Accurate: Sometimes – e.g. on certain web-pages – the form-filling function of a form-filler may not work correctly. All form-fillers are plagued with this problem which comes in four flavors:
   a. An input-field that should be filled (and input-data is available) will not get filled. Example: A membership-number field of a log-in form which should get filled-in with input-data of type 'username' will not get filled because it was not classified correctly.
   b. An input-field that should not be filled (and input-data is available) will get filled. Example: an email field of a contact or address form gets filled-in with username that is intended for a login form because it was misclassified as a username field and/or the form was classified as a log-in form.
   c. An input-field will get filled with the wrong type of data. E.g.: username field gets filled in with password because the DOM type attribute of the input-field had value 'password' (CSS: 'input[type=password]')
   d. Input-data that should have been applied to a web-page was not owing to incorrect identification/categorization of a web-page. For e.g. the login-id of facebook.com should be applied to pages with hostname www.facebook.com as well as apps.facebook.com because the same login credentials apply to both hostnames. However, this aspect is out of the scope of this document and is being mentioned here only for completeness sake.

Definition: The ability of a form-filler to fill-in input-fields shall be called 'automatic form filling' or 'auto-fill' in this document.

As described above, password-manager and form-filler applications have varying levels of capabilities vis-à-vis auto-capture, auto-fill and recognition. This invention has four parts:

1. Part 1: A convenient manual way to fill user-input fields with previously stored input-data. Use the drag-and-drop user-interface gesture to fill-in input-fields either as a primary method of form-filling or as a way of working around auto-fill inaccuracies summarized in paragraph: Form Filling is not a 100% Accurate.
2. Part 2: Leverage the drag-and-drop form-filling method of Part 1 to unobtrusively infer and collect user feedback regarding form and input-field recognition. This will eventually help in categorizing user input fields into useful categories such as 'username', 'password' etc. and forms into similarly useful categories like 'sign-in form', 'sign-up form', 'address form' etc. Existing applications do one or more of the following:
    1. Do not attempt to categorize forms or input fields. Instead, just store the user input data along with form, field and webpage identifiers and re-fill it the next time the same field is seen on the same webpage.
    2. Categorize a form / input field and input-data after the input data has been captured – in some cases by asking the user to categorize it.
    3. Categorize a form or user-input field automatically using intelligent algorithms.

    The second part of this invention focuses on improving applications that do attempt to categorize forms or user-input fields. Part 2 of this invention is the leveraging of the drag-and-drop form-filling method of Part 1 to unobtrusively infer and collect user feedback regarding form and input-field recognition.

3. Part 3: Augment or build the form-filler's form and input-field recognition capability by incorporating the user-feedback of part 2 into the form-filler's recognition capability.
4. Part 4. Collect the user-feedback from all users of the application worldwide and incorporate it into future releases of the form-filler and any other applications that may benefit from it.

Invention Summary

This invention has four parts:

1. Part 1: The provision of drag-and-drop UI gesture to fill-in input-fields as a convenience for the end-user.
2. Part 2: A method for the form-filler to infer and collect user feedback regarding form and input-field recognition when the user performs drag-and-drop operations mentioned in part 1.
3. Part 3: Augment or build the form-filler's form and input-field recognition capability by incorporating the user-feedback into the form-filler's recognition capability.
4. Part 4: Collect the user-feedback from all or several users of the application worldwide and incorporate it back into the form-filler and any other applications that may benefit from it

Invention Description

The four parts of the invention are described next.

Part I: Drag and Drop Form Filling: A form-filler that needs to fill-in input-fields may make available the following manual yet convenient drag-and-drop based method for its users to fill-in input-fields. This method may be provided either as the primary auto-fill method of the form-filler, it may be provided as an alternative to other auto-fill methods or it may be provided as a workaround when the other methods fail as described in paragraph: Form Filling is not a 100% Accurate.

Drag-and-drop form filling

1. The form-filler presents a visual representation (i.e. user interface) of the input-data relevant to the context - the context could be a webpage or website or a standalone application. The category of each piece of input-data is known to the form-filler (e.g. ‘username’ and ‘password’ categories). The form-filler sets up the input-data’s UI elements (e.g. HTML5 DOM elements) in a state such that they are ready to be ‘dragged and dropped’. This state is called ‘draggable state’. One possible visual representation of UI is shown in Figure 2. The form filler will use the appropriate platform APIs to set up the UI elements into draggable state. For example, in HTML5 DOM API this would be accomplished by setting the DOM element’s property ‘draggable’ to the state ‘true’. Other platforms may provide other APIs – a full survey of which is out of scope of this document.

2. The user initiates a ‘drag and drop’ operation by following appropriate user-interface actions depending on the type of the human-computer-interface capabilities available.

   If the human-computer-interface was like that of a typical laptop or desktop with a mouse device, the user would move the mouse pointer to an input-data UI element that they wanted to fill into the misclassified or unclassified user-input-field. This would typically cause the mouse-pointer’s visual representation to change in order to hint to the user that the UI element was ‘draggable’. If the device was a ‘touch device’ such as a ‘tablet computer’ (e.g. Apple iPad™) or a mobile smartphone (e.g. Apple iPhone™) then appropriate ‘drag and drop’ - called ‘click and drag’ – procedure should be followed.

   Drag Data Store Contents
   Just as the drag operation begins, the form-filler shall use platform APIs to insert the following data items into the drag-data-store object which is provided by the platform API. This is an object that carries the drag-and-drop / click-and-drag data for a given instance of drag-and-drop/click-and-drag operation. Data inserted into this object by the ‘drag-source’ (term defined later) can be retrieved by the ‘drop-target’ (term defined later). If the platform was a HTML5 browser, then the drag-data-store is accessed through a (JavaScript) object bound to the dataTransfer property of the dragstart event. Please see HTML5’s Drag and Drop specification for reference. Other operating platforms such as Microsoft Windows and Mac OSX also provide their own equivalents of drag and drop (also called click and drag). The form-filler shall insert the following items of data into the drag-data-store object.

   a. The input-data,
   b. Input data’s field-category (e.g. ‘username’ and ‘password’ categories),
   c. Identifying information of the form-filler itself,
   d. Optionally, an image to show in place of the mouse pointer as the data is being dragged,
   e. Any other information it may need to insert for additional functionality

   The form-filler shall also setup the platform to allow copying data to a target input-field. For e.g. in HTML5 this would be done by setting the effectAllowed property of the dataTransfer object to ‘copy’.

   Definition: In this document, the input-data’s UI element shall be called ‘drag source’. The form-filler shall be called the drag-source-application.

3. The user would hold the primary mouse button down and start moving the mouse pointer toward the field where he/she intends to input / fill the data. At this point, the user is said to be ‘dragging’ the data. Please see Figure 3. Typically, the mouse-pointer will take on a different visual form – in order to indicate that the drag operation is in progress. If the device was a ‘touch device’ such as a ‘tablet computer’ (e.g. Apple iPad™) or a mobile smartphone (e.g. Apple iPhone™) then appropriate ‘drag and drop’ - called ‘click and drag’ – procedure should be followed.

4. With the mouse-pointer hovering over the intended target input-field, the user would then release the primary mouse button. This constitutes a ‘drop operation’. After the drop, the application where the drop operation occurred would fill the said input-field with the dragged input-data if it had the

capability to receive dragged data. The user would see that they had dragged the input-data and dropped it into the input-field. Please see Figure 4.

Definition: The input-field which receives the dragged data shall be called the 'drop-target' and the software application controlling it, the 'drop-target-application'.

Most visual UI applications support the dropping of text-data into their input-fields i.e. they have 'drop capability'. This is so because most operating environments support this behavior by default – i.e. the standalone application programmer or web application programmer would not need to make any effort in providing drop capability in their UI application. To the drop target application, this would appear as a regular input event and no special coding would be required to handle dropped data. Therefore, coding effort is only required on part of the drag source application – i.e. the form-filler where the drag operation started.

5. Please note that the drag-source (i.e. the input-data element) and the drop target (i.e. the input-field) could be in different web-pages, different browsers or even different applications. Figure 2 through Figure 4 depict drag-and-drop operation within the same browser web-page. On the other hand Figure 5 through Figure 7 depict drag-and-drop operation from an application running inside a browser web-page to a different application running outside the web browser.

   Definition: The user actions described in this section (**Drag-and-drop form filling**) shall be called 'drag-and-drop operation' or 'dragging and dropping'.

As stated above, more than one application may be involved in the end-to-end drag-and-drop operation. The drag operation may begin within one application (e.g. a form-filler application running within a web browser page or running as a standalone desktop, tablet or mobile application), then the mouse pointer may be dragged over the visual user-interface of one or more applications (e.g. the mouse pointer may be dragged across the operating system's desktop or across another application's window), and finally dropped into an input field in yet another application. Of course, the entire process could also start and end within the same application or within the same web-browser (but different pages) or within the same web-page of the same web-browser. Note however, that even in case the entire drag-and-drop operation commenced and ended within the same web-browser page, the sending and receiving applications could still be different; for e.g. the drag-source-application could be a web-browser extension's content-script running in a web-page while the drop-target maybe an input-field on the web-page originating from a different location. Conceptually though, the drag-and-drop operation is a collaboration between the drag-source-application, the drop-target-application and the operating environment which acts as a liaison between the other two and manages the drag-data-store. This is a common capability provided by most operating environments with visual UI.

The drag-and-drop or click-and-drag behavior is not being claimed by this patent application, rather the use of drag-and-drop by a form-filler is the innovation being claimed herein. This is an innovation because none of the existing form-filler applications utilize this technique to work around significant deficiencies in their auto-fill and auto-capture capabilities as explained earlier.

Part 2: Inferring User Feedback from Drag and Drop

If the drop-target was under control of the form-filler application then the form-filler could infer feedback to its field recognition algorithms as follows:

1. The form-filler shall setup 'drop' event handlers ('drop handler' for short) on all input-fields in its control. In HTML5 DOM, this is done by adding an event listener on all input-field elements (these were already described in the Background section) against the 'drop' event.
2. When a drop event is received, the drop-handler function of the form-filler shall be invoked. The form-filler shall then inspect the contents of the drag-data-store that was created at the drag-source. If the drag-data-store was created by the form-filler application, then it will have all the data in the format described in **Drag Data Store Contents**. The form-filler shall look for the drag-source identifying

information in the drag-data-store. If it finds it and it identifies the form-filler application, then it means that the drag-source-application is the form filler application - either the same instance as the drop-target-application or a different instance of the form-filler software. It shall proceed to the next step if the data-source was setup by the form-filler application itself (either same or another instance) or by another compatible application.

3. As stated earlier, the form-filler may have already categorized the input-field based on some heuristic/AI algorithms, prior knowledge of the web-page etc. Now the form-filler shall retrieve the input-field category from the drag-data-source and compare that with the category that it has determined so far – if any. If the two categories are different, then it means that the existing categorization was incorrect, because presumably the user-action is more accurate. That is, by dropping data of a given field-category into the input-field, the user has indirectly categorized the input-field. This is user feedback that the form-filler shall store for future reference. In order to ensure that spurious or accidental drag-and-drop actions won't get construed as user-feedback, the form-filler may require the user to perform an additional action in order to confirm the user-feedback. For example it may require the user to hold down the control key while dragging and dropping the data. The form-filler shall store the following pieces of information in its persistent data-store as part of this method:
    a. Input-field-category obtained from the drag-data-store,
    b. As much identifying information as it can get, of the input-field element as already described in paragraph: **Field Identifying Information**. For example – in case of a HTML element – its tag-name, attributes: 'name', 'id', 'type' and 'name' and 'id' attributes of its associated form or container element.
    c. Identifying information of the web-page or UI-page of the application containing the drop-target,
    d. The current data and time and
    e. Any other information it may need for additional functionality

    Definition: Each instance of user-feedback thus inferred and stored shall be called a knowledge-record henceforth in this document. The total collection of knowledge-records shall be called knowledge-db.

4. Newer instances of knowledge-records of a given input-field on a given web-page/application shall override preceding ones. With continued user-feedback, the form-filler shall thus accumulate knowledge-records in its knowledge-db.

The next section will detail how the knowledge-db will be used to improve input-field categorization.

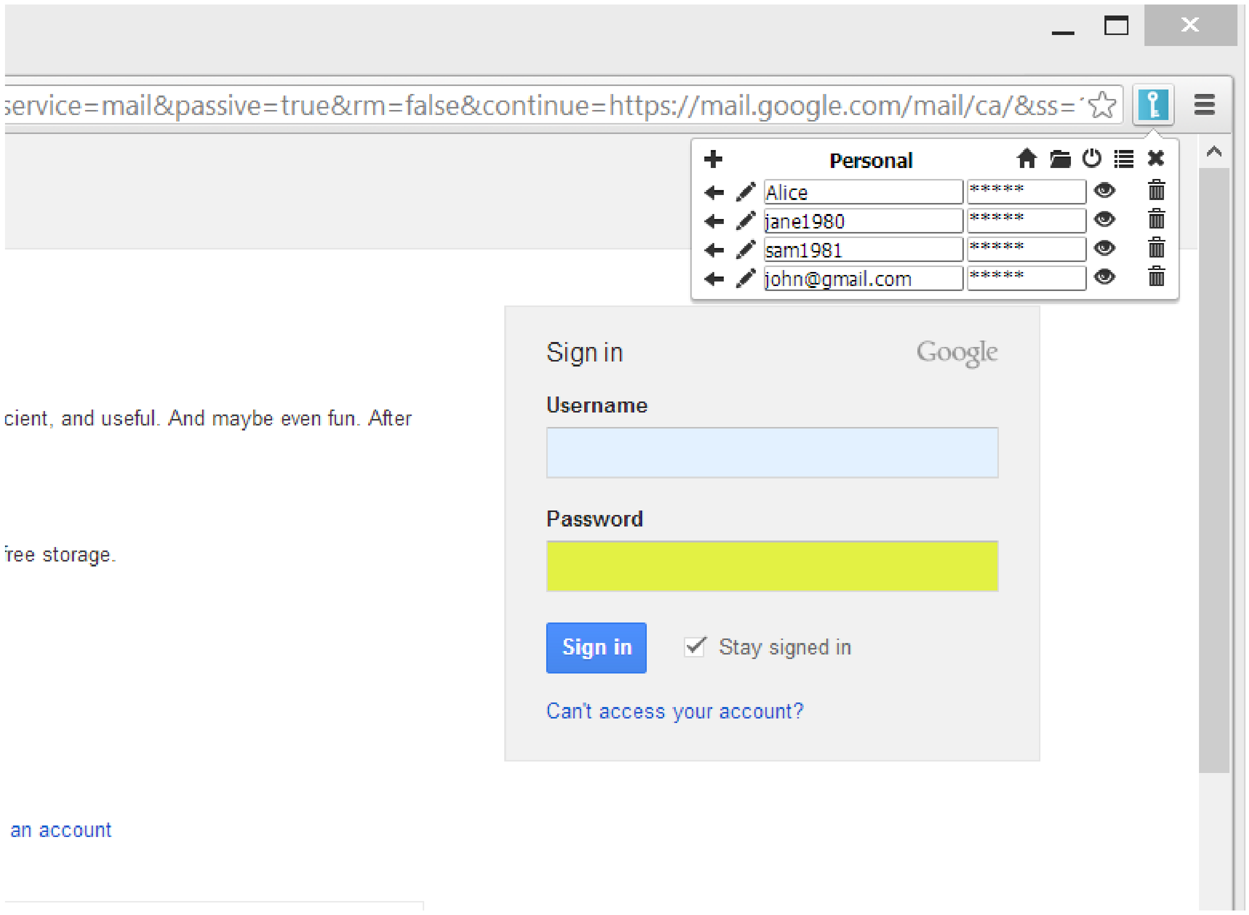
Part 3: Using knowledge-db for field-recognition

1. When the form-filler needs to recognize fields on a web-page or UI-page of a standalone application, it shall first retrieve all knowledge-records (from its persistent data-store) pertaining to the web-page or UI-page.
2. If any knowledge-records are retrieved, then it shall deem each field identified in a knowledge-record as belonging to the category indicated therein. No further processing is required for those input-fields.
3. Thereafter it shall run any other methods of field and form categorization for the web-page or the UI-page – e.g. any heuristic and AI algorithms it may have. These categorization methods are expected to only be used to categorize any remaining fields and forms on the page, because the categorization found in the knowledge-records is deemed to be authoritative. However, a specific implementation may choose to use the knowledge-records as hints rather than as authoritative records and incorporate the hints into a broader categorization methodology.

Part 4: Crowd Sourcing and Continously Integrating Knowledge-DB

Part 2 details a way to create and store knowledge-records in a persistent data-store that the form-filler will look up later. This section extends that idea to collecting knowledge-records from several or all instances of the form-filler application deployed worldwide and incorporating them back into several or all form-filler instances deployed worldwide. This way users will benefit from each other's knowledge. The process is specifically as follows:

1. The form-filler periodically uploads new knowledge-records to a server on the internet.
2. The server may accept those knowledge-records right away or alternatively, send them off for manual inspection. After the knowledge records pass inspection, they are inserted into the knowledge-db maintained by server.
3. Any new deployments of the form-filler will inherit the knowledge-db maintained by the server.
4. Existing deployments of the form-filler will periodically download new knowledge-records from the server and incorporate those into their own respective knowledge-dbs.



ervice=mail&passive=true&rm=false&continue=https://mail.google.com/mail/ca/&ss=

Alice	******
jane1980	******
sam1981	******
john@gmail.com	******

cient, and useful. And maybe even fun. After

ree storage.

Figure 1 Example of a password-manager application that has scanned a web-page and categorized the input fields as username (blue background) and password (yellow background). It has also retrieved input-data (usernames and passwords) that are pertinent to the web-site and has presented it as rows of username and password records (the box in the upper-right corner). Each of the data-fields is draggable.

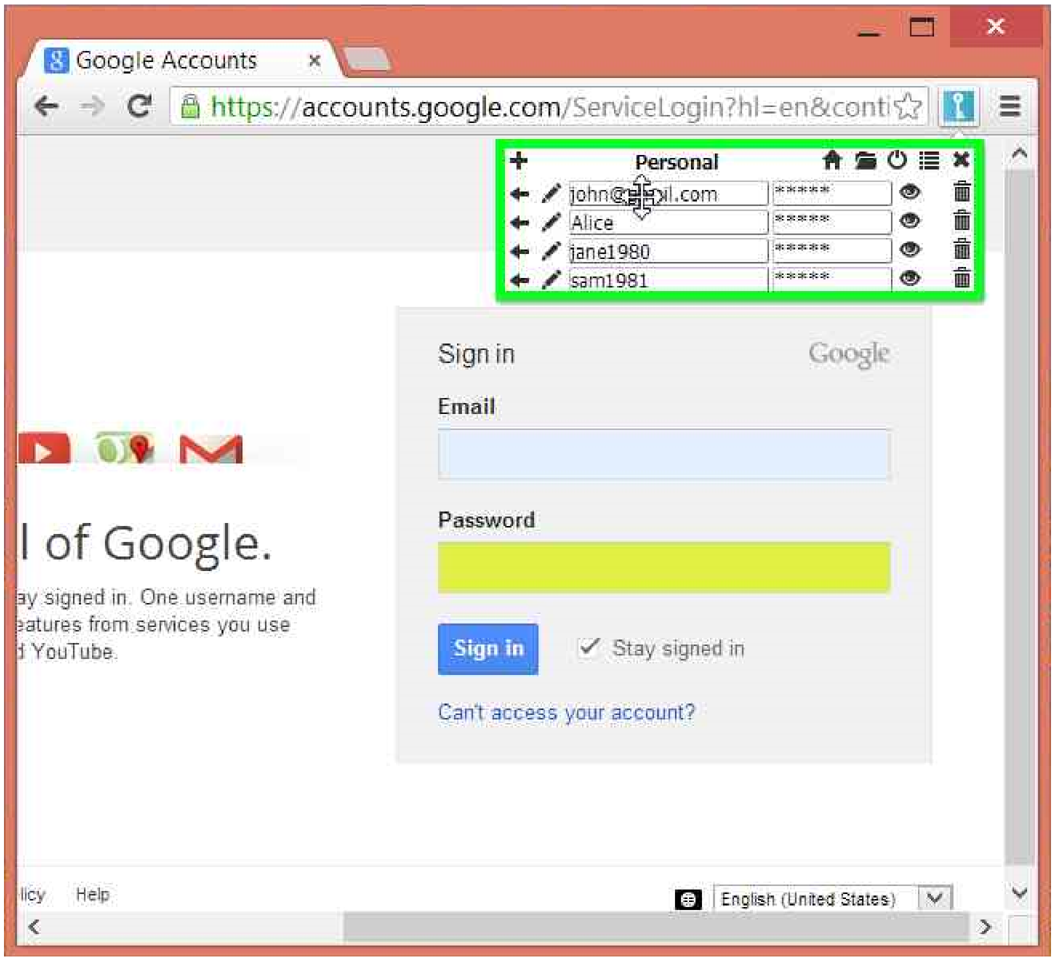


Figure 2 Drag-and-Drop Step 1. The password manager displays a set of username and password fields that are applicable to the webpage the user is on. The user moves the mouse pointer to the top of the username field he/she wants to fill into the 'Email' input field below, holds down the primary mouse button and starts dragging the mouse pointer toward the 'Email' field below

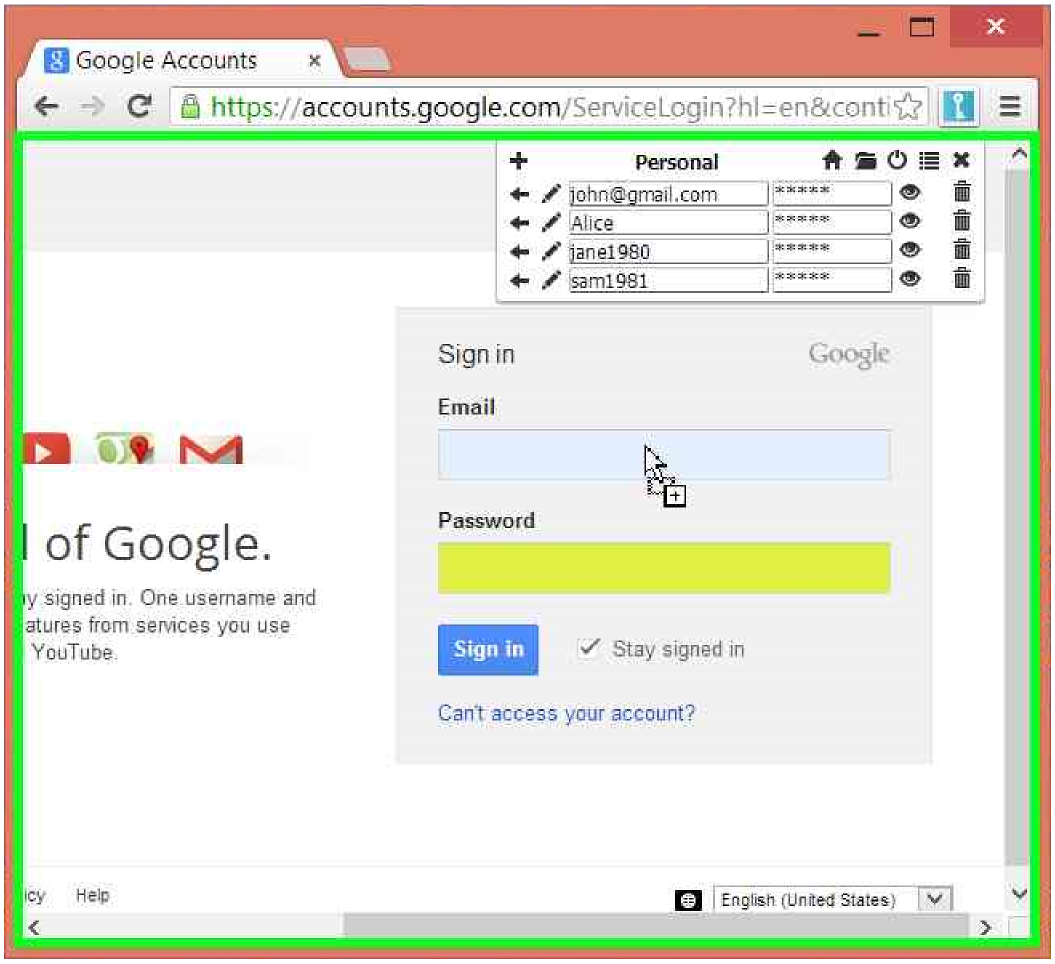


Figure 3 Drag-and-Drop Step 2. The user drags the mouse pointer on to the 'Email' field and gets ready to drop the dragged input-data onto the field.

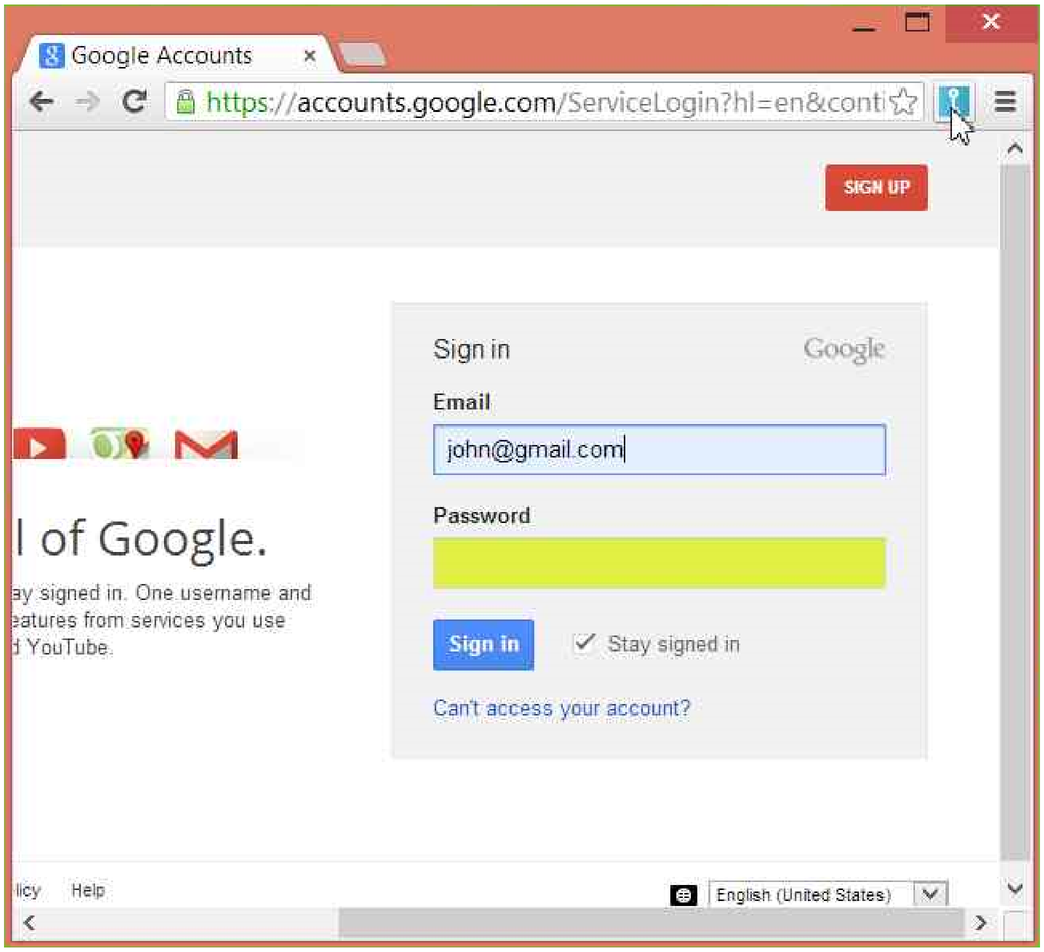

Google Accounts
https://accounts.google.com/ServiceLogin?hl=en&conti
SIGN UP
Sign in
Google
Email
john@gmail.com
Password
Sign in
Stay signed in
Can't access your account?
l of Google.
ay signed in. One username and
eatures from services you use
d YouTube.
licy
Help
English (United States)

Figure 4 Drag-and-Drop Step 3. After dragging the mouse pointer to the Email field, the user releases the primary mouse button. This causes the application to insert the dragged input-data into the Email field and categorize the Email field as 'Username' category.

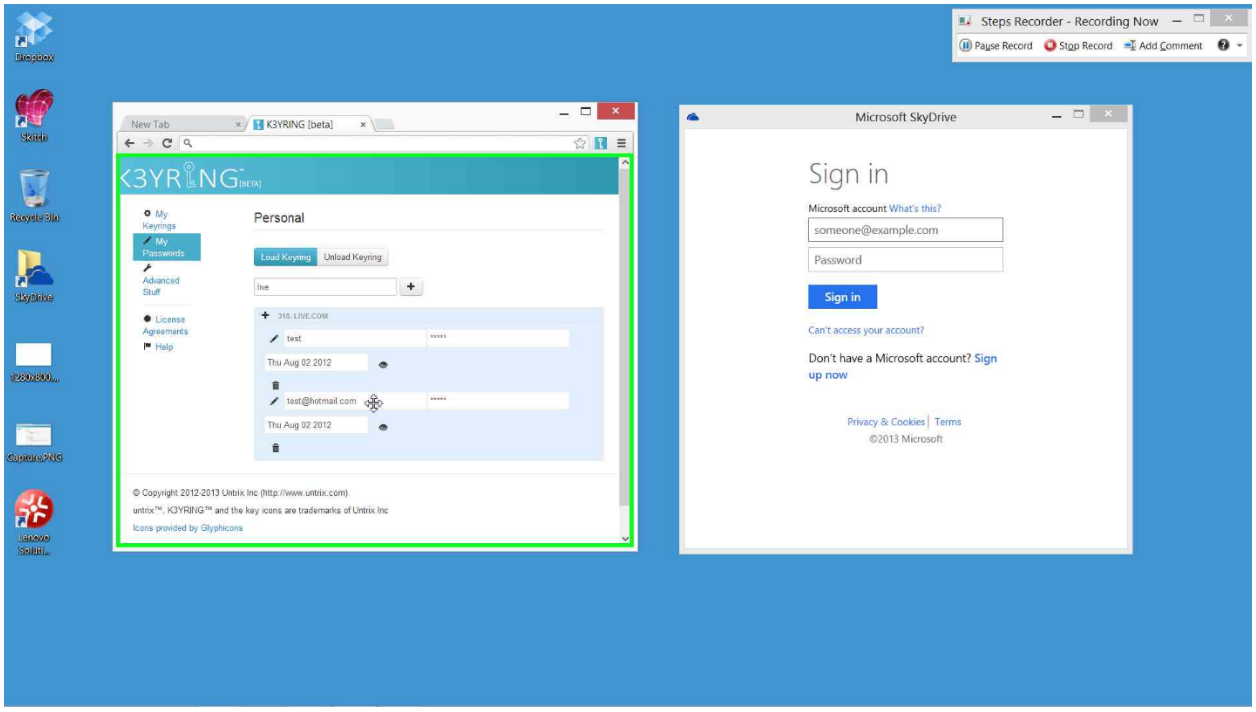


Figure 5 Drag and Drop from Web Browser to a different application. Step1: Begin drag

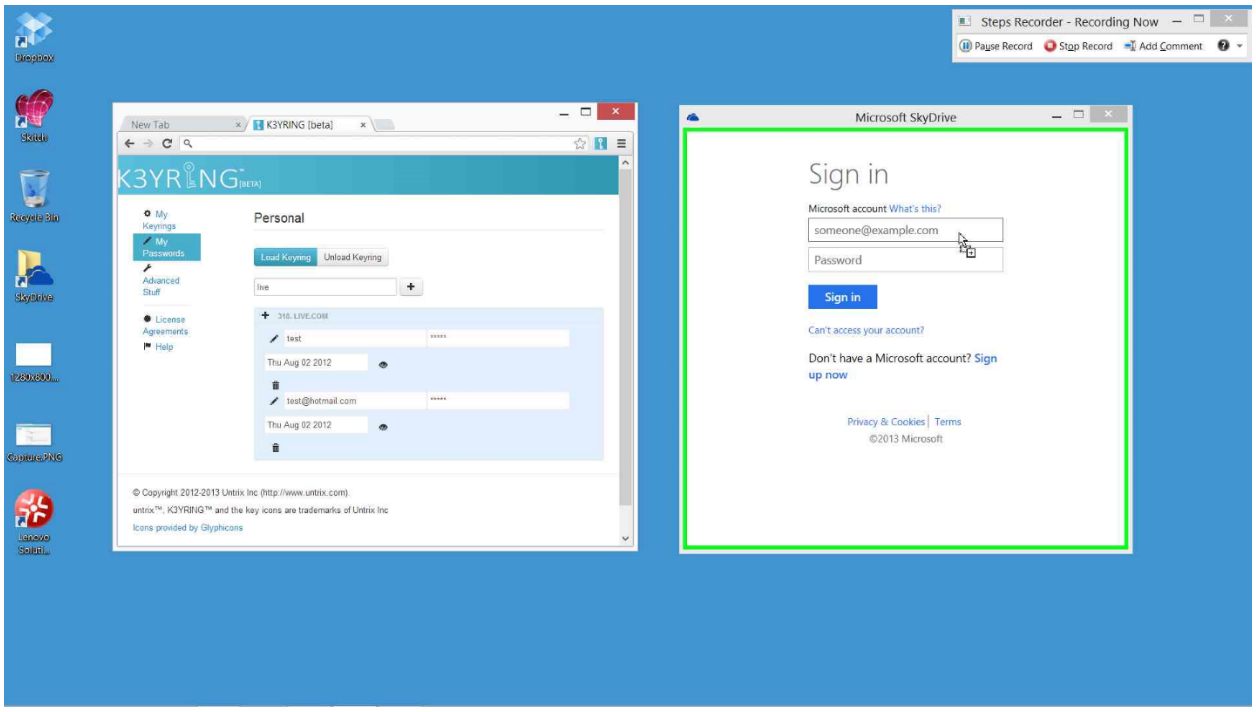
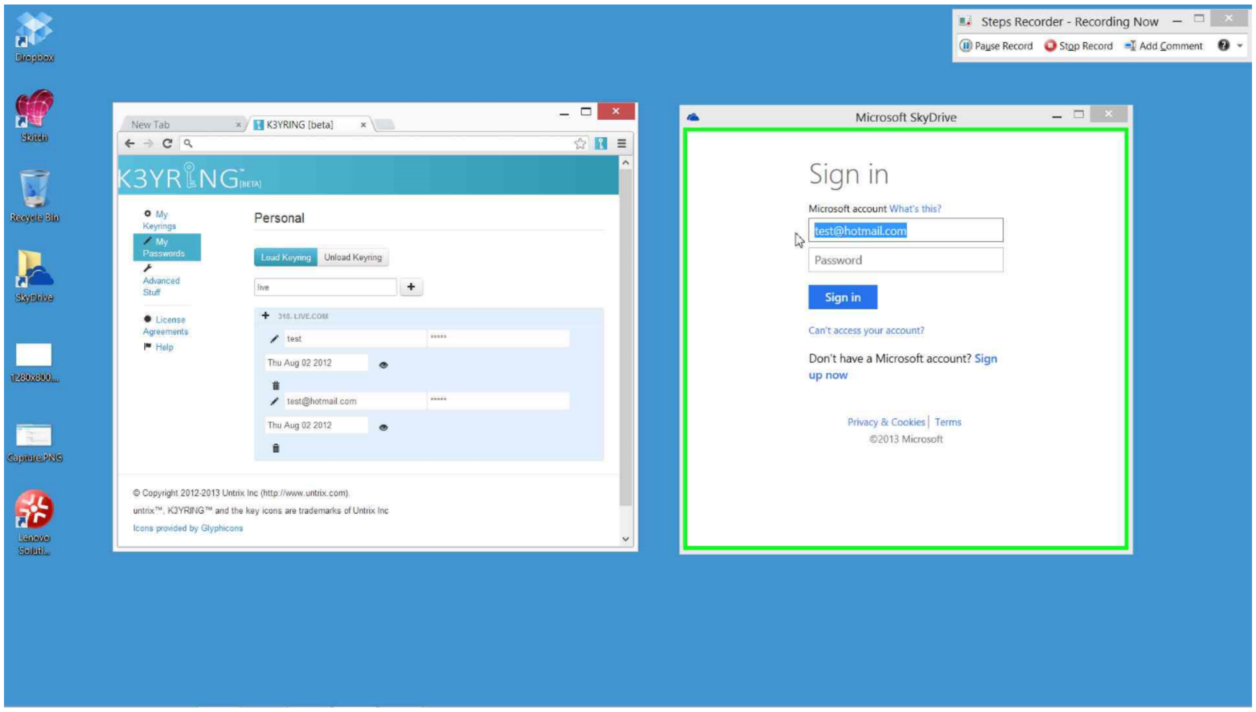


Figure 6 Drag and Drop from Web Browser to a different application. Step2: Just before the drop operation



Personal

live

318. LIVE.COM

test *****

Thu Aug 02 2012

test@hotmail.com *****

Thu Aug 02 2012

© Copyright 2012-2013 Untrix Inc (http://www.untrix.com).

untrix™, K3YRING™ and the key icons are trademarks of Untrix Inc

Icons provided by Glyphicons

Microsoft account What's this?

test@hotmail.com

Password

Sign in

Can't access your account?

Don't have a Microsoft account? Sign up now

©2013 Microsoft

Figure 7 Drag and Drop from Web Browser to a different application. Step3: Just after the drop operation