Intro to libYUI

Agenda

- what's libYUI ?
- small demo
- usage pattern in openQA testing
- QA time

andrea.manzini@suse.com

What's libYUI ?

Libyui is a widget abstraction library providing graphical (Qt) and text-based (NCurses) front-ends, developed on https://github.com/libyui/libyui

Its purpose is to provide an abstraction layer that permits to write user interactive code without knowning the underlying implementation (ncurses, QT, GTK)

For our testing purposes, we'll focus only on a specific subset of the API:

`libYUI-REST-API`

What's libYUI-REST-API ?

The solution allows to query the UI properties over HTTP using a REST API. This allows to automate the UI interaction steps and avoid screen-based tools. The API allows reading properties of the UI, so the displayed values can be validated. It also allows interacting with the UI (clicking buttons, toggling check boxes, entering text, etc.).

NOTE: libyui-rest-api is only available starting from SLE 15 SP3.

documentation on usage

can I use it even from YaST installer ?

Yes! Add the parameters in the boot cmdline

`extend=libyui-rest-api YUI_HTTP_PORT=39122 YUI_HTTP_REMOTE=1 YUI_REUSE_PORT=1`

then you can "interact" with YaST installer via simple HTTP GET/POST requests.

`curl http://YOUR_IP:YOUR_PORT/v1/application`

`curl http://<vm_ip>:39122/v1/widgets`

- github API docs
- confluence

Tips

- you can get the ip of the virtual machine from virt-manager "virtual machine details" or with `sudo virsh net-dhcp-leases --network default`
- you can specify any port to bind, given it's free. Better to choose a random one
- redirect json output in order to save it as reference

DEMO TIME

- start installation
- type at the boot prompt `extend=libyui-rest-api YUI_HTTP_PORT=9999 YUI_HTTP_REMOTE=1 YUI_REUSE_PORT=1`
- start installation
- `curl http://<vm_ip>:9999/v1/application`

use of libYUI-REST in openQA

- actual API calls are wrapped in YuiRestClient.pm, so no need to fiddle with http
- testsuite needs to include `setup_libyui.pm`
- `YUI_REST_API` variable flag needs to be set
- base class is `'y2_installbase'`
- design pattern is a variant of MVC : Page-Controller
- test module itself if very simple and straight, often only a few lines:
  - instantiate a new controller obj
  - invoke methods to send command actions to the controller
    - the controller accesses to the proper page obj
      - the page object act on the page widgets

test module in practice

… from this …to this:

installation/product_selection/install_SLES.pm

```perl
use base 'y2_installbase';
use strict;
use warnings;

sub run {
    $testapi::distri->get_product_selection()->install_product('SLES');
}

1;
```

controller creation

`get_product_selection()` is a function defined in `lib/Distribution/*` that returns a new Controller object. Using the `$distri` abstraction, each distribution can override and implement it in a different way, or fallback to the default.

```
sub get_product_selection {
    return Installation::ProductSelection::ProductSelectionController->new();
}
```

controller inner working

- a Controller like ProductSelectionController is a Perl class that contains every 'page' under its domain. For example popups, buttons and list of sub-items.
- From the test module we **always and only access and call Controller methods**.
- as a rule of thumb, controller methods should be high-level actions, like "install this product" or "register with this code", demanding actions to the Page methods

page inner working

- a Page like ProductSelectionPage is a Perl class that contains the set of elements (buttons, radio, labels …) and is called by the controller to perform actions ("click this button", "fill this field with this text") or to get informations like "is that popup being shown" ?

you can see more examples in openQA like this or this

Conclusions

usage of libYUI-REST-API has many benefits:

- no more fiddling with needles
- tests runs [or fails] much faster, no more delays or waiting
- code is straightforward to write and follow/debug
- tests are independent about text or graphical interface

cons:

- more code modules to write, need to get used to Perl OOP quirks
- need some knowledge of libYUI widget hierarchy and IDs

Questions ?