e-terface Technical Documentation v1.0

Table of Contents

What is e-terface

e-terface's purpose is to provide an alternative graphical user interface (GUI) for modern operating systems. Instead of relying on existing operating system (OS) application programming interfaces (API) or frameworks (such as QT), the power of HTML and other web development languages may be used. Much work has been put into making web pages visually appealing to wide audiences. One of these reasons is to increase the conversion rate, and is vital for companies who want to increase their sales online. The idea here is to give web developers who have a background in design, a chance to be able to take their talents to OS GUI design. Web technology is advancing every day, and so it should offer powerful visualization opportunities. Imagine some of your favorite sites, and imagine being able to navigate an OS in a similar manner.

By using web technologies instead of C++ for example, it is hoped that e-terface could make OS GUI design simpler.

The name "e-terface" is a play on e-mail and e-commerce. The

interface is delivered via the web, instead of being part of the OS itself.

High-level overview

The e-terface follows the client/server model. The client will visualize all of the data that it receives from the server, while the server will manage user interactions with the host system. The server provides a web service for the client to use to visualize the desktop environment. Although we refer to the "server", there are really two main modules: visualization and service modules. The visualization module provides data and state services to the client for visualization purposes. The service module provides actual system-related services for the client such as file read/write. Because the GUI is separate from the OS itself, the GUI will need to provide some applications that already exist on the home server (such as a text editor or even a console). In the case of a text editor, the service module will receive a write request from the client containing the new data, and write that to the appropriate file. Since this is version 1.0 of both the document and the server, e-terface will be fairly limited to simple functions.

To write a client, the following steps must be followed:

1. Retrieve the list of users from the server by calling /login/userList.

2. The usersList request will return a JSON object with an array containing all users on that system. A user must select the proper user entry.

3. Once the user is selected, send this to the server using /viz/initUser. This will add the user, and loads their entire home directory (including its sub directories) into the server. Once the directory is loaded, the server will serialize just the home directory and return it as JSON.

See the sections: Services API and Appendix for more details.

Technology

Client/interface:
The client is supposed to run in a web browser, and visualize desktop data using web technologies. At least one of JavaScript (XMLHttpRequest) or ActionScript 3 (URLRequest) will be needed to visualize the desktop. Of course, Java or C++ or some other technology could also be used to visualize the desktop.

Server:
Java EE will be the programming language used to implement the e-terface server. The web container will be Apache Tomcat 7 since the project is open source. Tomcat 7 should use a JRE that is at least version 7.

Host environment:
Right now, e-terface supports Linux and Windows environments.

Data format:
The data exchange format will be JSON, at least for the time being. See the appendix for example JSON responses.

Installation instructions

This section is pretty limited for now. For the most part, a Java EE Eclipse IDE, Tomcat 7 installation, and JRE that is at least version 7 are required. Setting up e-terface requires a dynamic web project in Eclipse. Included in WebContent/WEB_INF/lib should be a javax.json JAR, which is used in some classes. It may be necessary to add that JAR to your project by going to "project->properties->java build path". Here are some links for setting these things up:

Latest version of Java EE Eclipse IDE:

https://eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/lunasr2

Tomcat 7 download:

https://tomcat.apache.org/download-70.cgi

Adding a Tomcat installation to Eclipse:

Note: replace the "Installed Runtimes" part with "Runtime Environments" if your version does not include that.

http://www.eclipse.org/webtools/jst/components/ws/1.5/tutorials/InstallTomcat/InstallTomcat.html

Services API

Service Method	Method Description	Method Params
/viz/dir/peek	Returns the contents of some directory.	path={$val} user={$val}
/login/userList	Gets the list of users in the system	
/viz/initUser	Adds the selected user to the server, and returns the json serialization of their home/desktop.	user={$val}
/res/getResource	Retrieves the given resource from the server/ Machine (does not return JSON).	path={$val}

Example API calls (taken from ServicesServlet.java):

initUser:
http://localhost:8080/e-terface/viz/initUser?user=myuser

userList:
http://localhost:8080/e-terface/login/userList

peek:
http://localhost:8080/e-terface/viz/dir/peek?
user=myuser&dir=/usr/home/myuser/Desktop

getResource:
http://localhost:8080/e-terface/res/getResource?
path=/usr/home/myuser/res.ext

Visualization module

The visualization module maintains user directory state, and provides any kind of visualization service. For now, all it does is maintain state, but it could perhaps maintain a configuration directory for each user. This configuration directory could save things like themes, or background images, and so on.

Services module

The services module provides system-related services for the GUI. For example, if a user wants to create a directory, or write the contents of a text file. The services module will also be able to execute some command line programs on the user's behalf. For now, this module does nothing, and only exists as a stub.

Appendix

Example JSON from /viz/dir/peek and /viz/initUser:

```
{
      "directoryName": "value",
      "directoryPath": "/dir/value",
      "numberOfItems": 4,
      "directoryList": [
                              {
                                    "directoryName": "value1",
                                    "directoryPath": "/dir/value/value1",
                                    "numberOfItems": 1
                              },
                              {
                                    "directoryName": "value2",
                                    "directoryPath": "/dir/value/value2",
                                    "numberOfItems": 1
                              }
                           ],
      "fileList": [
                        {
                              "fileName": "value1.txt",
                              "fileSize": 1,
                              "fileType": "txt",
                              "filePath": "/dir/value/value1.txt"
                        },
                        {
                              "fileName": "value2.txt",
                              "fileSize": 1,
                              "fileType": "txt",
                              "filePath": "/dir/value/value2.txt"
                        }
                     ]
  }
```

Example JSON from /login/userList:

```
{ "userList": ["userOne", "userTwo", "userThree"] }
```

List of JSON error codes returned and description:

ERR_PATH_NOT_FOUND : Incorrect path given in parameter

ERR_URL_DECODE : Issue with character encoding in the URL (server uses ASCII at the moment).

ERR_NO_SUCH_METHOD : Attempted to call a method that doesn't exist.

ERR_FILE_IRRETRIEVABLE : Requested a resource that is either in-use by the system, privelige issue, or some other reason. See

http://docs.oracle.com/javase/7/docs/api/java/nio/file/FileSystemException.html

See next page for UML diagram/high-level overview

+ EterfaceDirectory
-fileList : ArrayList<String>
-directoryList : ArrayList<EterfaceDirectory>
-directoryPath : String
-directoryPath : String
-numberOfItems : Integer
+getNumberOfItems() : Integer {sequential}
+getDirectoryName() : String {sequential}
+getDirectoryPath() : String {sequential}
+getFileList() : ArrayList<File> {sequential}
+getDirectoryList() : ArrayList<EterfaceDirectory> {sequential}
+getDirectoryLocal() : EterfaceDirectory {sequential}
+getDirectoryAbsolute() : EterfaceDirectory {sequential}
+setNumberOfItems(size : Integer) {sequential}
+setDirectoryName(name : String) {sequential}
+addDirectory(directory : EterfaceDirectory) : Boolean {sequential}
+addFile(file : File) : Boolean {sequential}
+serialize() : String {sequential}
+ EterfaceHome
-userName : String
-homePath : String
+getUserName() : String {sequential}
+getHomePath() : String {sequential}
-fillDirectory(directory : EterfaceDirectory) {sequential}
0..*
1
+ VisualizationModule
-usersList : ArrayList<String>
-ERROR_PATH_NOT_FOUND : String
+initUser(user : String) {sequential}
+peek(user : String,path : String) : String {sequential}
+ ServicesServlet
-ERROR_URL_DECODE : String
-ERROR_NO_SUCH_METHOD : String
-ERROR_FILE_IRRETRIEVABLE : String
+ ServicesModule
+ SecurityModule
+ LoginModule