

EASYINTERFACE Tutorial

<http://abs-models.org/your-tutorial/>

JESÚS DOMÉNECH
SAMIR GENAIM



<http://www.envisage-project.eu/>

Introduction

TODO: Samir will add

Contents

1	General Overview	3
1.1	The Architecture of EASYINTERFACE	3
1.2	The Server Side	3
1.3	The Client Side	4
2	Step by Step Example	5
2.1	Add Your First Application to the EASYINTERFACE Server	6
2.2	Passing Input Files to Your Application	7
2.3	Passing Outline Entities to Your Application	8
2.4	Passing Parameters to Your Application	9
2.5	Using the EASYINTERFACE Output Language in Your Application	11
2.5.1	Printing in the Console Area	12
2.5.2	Adding Markers	13
2.5.3	Highlighting Code Lines	14
2.5.4	Adding Inline Markers	14
2.6	Opening a Dialog Box	15
2.7	Adding Code Line Actions	15
2.8	Adding OnClick Actions	16
3	The Envisage Setting	17
4	Further Reading	17

1 General Overview

The EASYINTERFACE framework provides a simple way to build interfaces, e.g., a web-interface or an Eclipse plugin, for tools written in (almost) any programming language. Moreover, it does not require the programmer to be familiar with any GUI library or web programming. Roughly, the only requirement is that the application can be executed from a command-line and that its output goes to the standard output.

The goal of EASYINTERFACE is to provide developers with a toolkit to *build their applications once and get several interfaces for free*. EASYINTERFACE was originally developed for building a common frontend for program analysis tools developed in the Envisage¹ project. This is why, as the reader will notice later, its graphical user interfaces are basically developing environments that allow editing programs, etc.

In the rest of this chapter we overview the different components of EASYINTERFACE, and explain how they are combined to achieve the above goal.

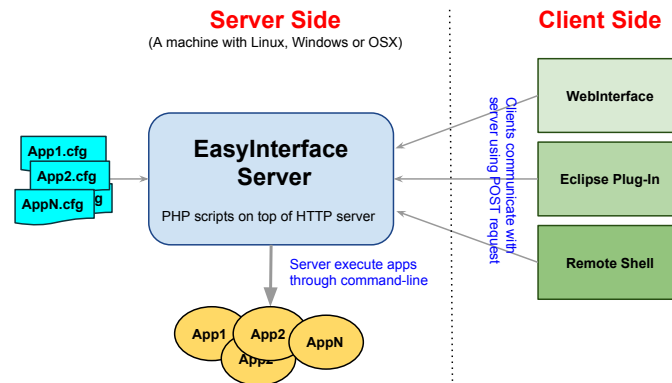


Figure 1: The Architecture of the EASYINTERFACE Framework

1.1 The Architecture of EASYINTERFACE

The architecture of EASYINTERFACE is depicted in Figure 1. It includes two main components: (1) *server side*: a machine with several applications (the circles App1, App2, etc., in Figure 1) that can be executed from a command-line and their output goes to the standard output. These are the applications that we want to make available for the outside world, i.e., execute them as services on the internet; and (2) *client side*: several clients that make it easy to communicate with the server side to execute an application, etc. In what follows, we first explain the inner components of the server side and which problems they solve, and then we explain the client side.

1.2 The Server Side

The problem that we want to solve at the server side is:

Provide a uniform way for remotely accessing locally installed applications as services.

This problem is solved by the EASYINTERFACE server, which is collection of PHP programs that run on top of an HTTP server. This server allows specifying how a local application can be executed and which parameters it takes using simple configuration files (App1.cfg, App2.cfg, etc., Figure 1). For example, the following is a snippet of such configuration file:

```
<app id="myapp" visible="true">
  ...
  <execinfo method="cmdline">
    <cmdlineapp>/path-to/myapp.sh _ei_parameters </cmdlineapp>
```

¹<http://www.envisage-project.eu>

```

</execinfo>
<parameters prefix = "-" check="false">
  ...
  <selectone name="c">
    <option value="1" />
    <option value="2" />
  </selectone>
</parameters>
</app>

```

This XML defines an application that has a unique identifier `myapp`. The `cmdlineapp` tag is a template that describes how to execute the application from a command-line. Here `_ei_parameters` is a template parameter that will be replaced by an appropriate value. The `parameters` tag includes a list of parameters accepted by the application. For example, there is a parameter called "c" that can take one of the values 1 or 2. Once the configuration file is installed on the EASYINTERFACE server, anyone can access the application using an HTTP POST request that includes the following text:

```

{
  command: "execute",
  app_id: "myapp",
  parameters: {
    c: ["1"],
    ...
  },
  ...
}

```

When the EASYINTERFACE server receives such a request, it generates a corresponding command-line (according to what is specified in the configuration file), executes it, and redirect the standard output back to the client.

1.3 The Client Side

Although we now have a relatively easy way to execute applications on the server side, it is still not as easy as we aimed at.

Our aim is to simplify this process further by providing (graphical) user interfaces that automatically (1) connect to the EASYINTERFACE server and ask for the list of available applications; (2) let the user choose an application to execute and set the values of the corresponding parameters; (3) generate a corresponding request and send it to the EASYINTERFACE server; and (4) shows the returned output to the user. The EASYINTERFACE framework provides three such interfaces: a *web-interface* that can be executed in a browser and looks like a developing environment (see Figure 2); an Eclipse-plugin that runs within the Eclipse IDE; and a remote-shell that can be used from a command-line.

Since the web-client and the Eclipse plugin are GUI based developing environments, EASYINTERFACE provide also, to an application, the possibility to generate output that has some graphical effects, e.g., open dialog-boxes, highlight code lines, add markers, etc. To use this feature, the applications should be modified to use the EASYINTERFACE output language. The following is a snippet of such output:

```

<highlightlines dest="/Examples_1/iterative/sum.s">
  <lines> <line from="5" to="10"/> </lines>
</highlightlines>
...
<oncodelineclick dest="/Examples_1/iterative/sum.c" outclass="info" >
  <lines><line from="17" /></lines>
  <eicommands>
    <dialogbox boxtitle="Hey!">
      <content format="text">
        Click on the marker again to close this window

```

</content>
</dialogbox>
</eicommands>
</oncodeclick>

The **highlightlines** indicates that lines 5–10 of the file `/Examples_1/iterative/sum.s` (which is opened in the editor) should be highlighted. The **oncodeclick** tag indicates that when clicking on line 17, a dialog-box with a corresponding message should be opened. Note that the application is only modified once to produce such output, and will have similar effect in all interfaces that support this output language.

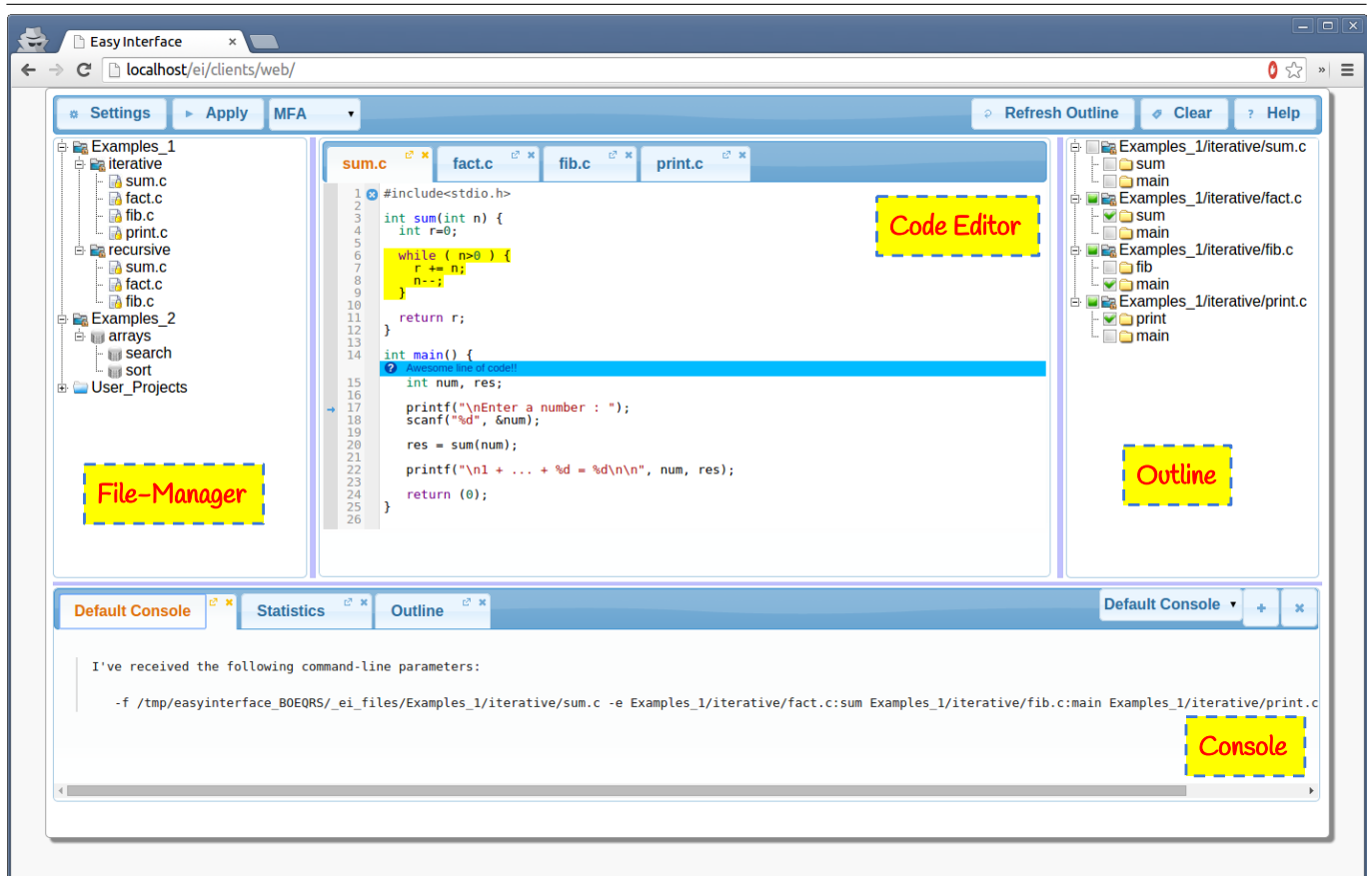


Figure 2: EASYINTERFACE Web Client

2 Step by Step Example

This section provides a quick introduction on *how to integrate an application in the EASYINTERFACE framework*. In particular, we develop a simple application, integrate it in the EASYINTERFACE server, and try it out through the web-client. The presentation is incremental, we start with a simple application and in each step we add more features to demonstrate the different parts of the EASYINTERFACE framework. In our explanation we assume that a Unix based operating system is used, however, we comment on how to do the analog operations on Windows when they are different. Note that in this section we only use the web-client (2), for other clients refer to EASYINTERFACE manual in <https://github.com/abstools/easyinterface>. We assume that EASYINTERFACE is already installed and working, which can be done following the instructions available in `INSTALL.md`.

Let us start by trying some demo applications that are available by default in the web-client. If you visit `http://localhost/ei/clients/web`, you should get a page similar to the one shown in Figure 2 on Page 5. At the top part of this page you can see a button with the label `Apply`, and to its right a

combo-box with several items Test-0, Test-1, etc. These items correspond to applications available in the web-client, and we will refer to it as the applications menu. To the left of **Apply** there is a button with the label **Settings**, if you click it you will see that each Test-i has also some parameters that can be set to some values. Note that, by default, the web-client is configured to connect to the EASYINTERFACE server at `http://localhost/ei/server` and ask for all applications, together with their corresponding parameters, that are available at that server. Note also that application Test-i actually corresponds to the bash-script `server/bin/default/test-i.sh`, and that its configuration file is `server/config/default/test-i.cfg` (later you will understand the details of such configuration files).

If you select an application, from the combo-box, and click on **Apply**, the web-client sends a request to the server to execute this application. The request includes also the current values of the parameters (those in the settings section) and the file that is currently active in the code editor area. The server, in turn, executes the corresponding program, i.e., the bash-script `server/bin/default/test-i.sh` in this case, and redirects its output back to the web-client. The web-client will either print this output in the console area, or view it graphically if it uses the EASYINTERFACE output language. Execute the demo applications just to get an idea on which graphical output we are talking about (e.g., highlight text, markers). In the rest of this chapter we explain, step by step, how to add your own application to EASYINTERFACE.

2.1 Add Your First Application to the EASYINTERFACE Server

When we add an application to the EASYINTERFACE server it will appear automatically in the applications menu of the web-client (unless you have changed the configuration of the web-client already!). Let us add a simple "Hello World" application.

We start by creating a bash-script that represents the executable of our application (it could be any other executable). We will place this bash-script in the directory `server/bin/default` together with the `test-i.sh` scripts, however, this is not obligatory and it can be placed anywhere in the file system as far as the HTTP server can access it. Create a file `myapp.sh` in `server/bin/default` with the following content:

```
1 #!/bin/bash
2
3 echo "Hello World!"
```

As you can see, it is a simple program that prints "Hello World!" on the standard output. Later we will see how to pass input to this application and how to generate more sophisticated output. Change the permissions of `myapp.sh` by executing the following (on Windows this is typically not needed):

```
> chmod -R 755 myapp.sh
```

Execute `myapp.sh` (in a shell) to make sure that it works correctly before proceeding to the next step.

Next we will configure the server to recognize our application. Create a file `myapp.cfg` in the directory `server/config/default` with the following content (we could place this file anywhere under `server/config` not necessarily in `default`):

```
<app visible="true">
  <appinfo>
    <acronym>MFA</acronym>
    <title>My First Application</title>
    <desc>
      <short>A simple EI application</short>
      <long>A simple application that I've done using the EasyInterface Framework</long>
    </desc>
  </appinfo>
  <apphelp>
    <content format='html'>
      This is my first <b>EasyInterface</b> application!
    </content>
  </apphelp>
```

```
<execinfo>
  <commandlineapp>./default/myapp.sh</commandlineapp>
</execinfo>
</app>
```

Let us explain the meaning of the different elements of this configuration file. The `app` tag is used to declare an `EASYINTERFACE` application, and its `visible` attribute tells the server to list this application when someone asks for the list of installed applications. Changing this value to `"false"` will make the application "hidden" so only those who know its identifier can use it. The `appinfo` tag provides general information about the application, this will be used by the clients to show the application name, etc. The `apphelp` tag provides some usage information about the application, or simply provide a link to another page where such information can be found. The actual content goes inside the `content` tag, which is HTML as indicated by the `format` attribute (use `'text'` for plain text). The most important part is the `execinfo` tag, which provides information on how to execute the application. The text inside `commandlineapp` is interpreted as a command-line *template*, such that when the server is requested to execute the corresponding application it will simply execute this command-line and redirect its output back to the client. Later you will understand why we call it *template*. Note that before executing the script, the server changes the current directory to `server/bin` and thus the command-line should be relative to `server/bin`.

Next we add the above configuration file to the server. This is done by adding the following line to `server/config/default/apps.cfg` (inside the `apps` tag):

```
<app id="myapp" src="default/myapp.cfg" />
```

Here we tell the server that we want to install an application as defined in `myapp.cfg`, and we want to assign it the *unique* identifier `"myapp"`. This identifier will be mainly used by the server and the clients when they communicate, we are not going to use it anywhere else. Note that in `default/apps.cfg` we used `"default/myapp.cfg"` and not `"myapp.cfg"`. This is because the server looks for configuration files starting in `server/config`. Note also that the main configuration file of the `EASYINTERFACE` server is `server/config/eiserver.default.cfg`, and that `default/apps.cfg` is imported into that file (open `server/config/eiserver.default.cfg` to see this).

Let us test our application. Go back to the web-client and reload the page, you should see a new application named `MFA` in the applications menu. If you click on the `Help` button you will see the text provided inside the `apphelp` tag above. Select this application and click on the `Apply` button, the message "Hello World!" will be printed in the console area.

2.2 Passing Input Files to Your Application

Applications typically receive input files (e.g., programs) to process. You must have noticed that the web-client provides the possibility of creating and editing such files. In this section we explain how to pass these files, via the server, to our application when the `Apply` button is clicked.

When you click on the `Apply` button the web-client passes the currently opened file (i.e., the content of the active tab) to the server, and if you use the `Apply` option from the context menu of the file-manager (select an element from the files tree-view on the left, and use the mouse right-click to open the context menu) it passes all files in the corresponding sub-tree. What is left is to tell the server how to pass these files to our application. Let us assume that `myapp.sh` is prepared to receive input files as follows:

```
> myapp.sh -f file1.c file2.c file3.c
```

In order to tell the server to pass the input files (that were received from the client) to `myapp.sh`, open `myapp.cfg` and change the command-line template to the following:

```
<commandlineapp>./bin/default/myapp.sh -f _ei_files</commandlineapp>
```

When the server receives the files from the client, it stores them in a temporary directory, e.g., in `/tmp`, replaces `_ei_files` by the list of their names, and then execute the resulting command-line (this is why we call it *command-line template!*). It is important to note that only `_ei_files` changes in the above template, the rest remain the same. Thus, the parameter `-f` means nothing to the server, we could replace it by anything else or even completely remove it — that depends only on how our application is programmed to receive input files.

Let us now change `myapp.sh` to process the received files in some way, e.g., to print the number of lines in each file. For this, replace the content of `myapp.sh` by the following:

```
1 #!/bin/bash
2
3 . default/parse_params.sh
4
5 echo "I've received the following command-line parameters:"
6 echo ""
7 echo "  $@"
8
9 echo ""
10 echo "File statistics:"
11 echo ""
12 for f in $files
13 do
14     echo " - $f has " `wc -l $f | awk '{print $1}` "lines"
15 done
```

Let us explain the above code. At line 3 we executes an external bash-script to parse the command-line parameters, the details are not important and all you should know is that it leaves the list of files (that appear after `-f`) in the variable `files`. Lines 5-7 print the command-line parameters, just to give you an idea how the server called `myapp.sh`, and the loop at lines 12-15 traverses the list of files and prints the number of lines in each one.

Let us test our application. First run `myapp.sh` from a shell passing it some existing text files, just to check that it works correctly. Then go back to the web-client, reload the page, select MFA from the applications menu, open a file from the file-manager, and finally click the **Apply** button. Alternatively, you can also select an entry from the file-manager and choose **Apply** from its context menu. You should see the output of the application in the console area.

2.3 Passing Outline Entities to Your Application

In the web-client, the area on the right is called the outline area (see Figure 2 on Page 5). Since `EASYINTERFACE` was designed mainly for applications that process programs, e.g., program analysis tools, this area is typically dedicated for a tree-view of program entities, e.g., method names, class names, etc. The idea is that, in addition to the input files, the user will select some of these entities to indicate, for example, where the analysis should start from or which parts of the program to analyze, etc. Next we explain how we can pass these selected entities to an application.

By default the web-client is configured to work with C programs, and thus if you open such a program (from the file-manager) and then click on the **Refresh Outline** button, you will get a tree-view of this program entities, e.g., method names (if you use **Refresh Outline** from the context menu in the file-manager you will get a tree-view of program entities for all files in the sub-tree). Note that to generate this tree-view the web-client actually executes a “hidden” application that is installed on the server, namely `server/bin/default/coutline.sh`, but this is not relevant to our discussion now (see Section 4.1.3 in the `EASYINTERFACE` manual in <https://github.com/abstools/easyinterface> for more details). Note also that `coutline.sh` is limited and will not works perfectly for any C program: it simply looks for lines that start with `int` or `void` followed by something of the form `name(...)`. This script is provided just explain how an application that generates an outline is connected to the web-client (see Section 4.1.3 in the `EASYINTERFACE` manual for more details).

As in the case of input files, the web-client always passes the selected entities to the server when the **Apply** button is clicked, and it is our responsibility to indicate how these entities should be passed to our application. Let us assume that `myapp.sh` is prepared to receive entities using a “-e” parameter as follows:

```
> myapp.sh -f file1.c file2.c file3.c -e sum.c:main sum.c:sum
```

In order to tell the server to pass the entities (that were received from the client) to our application, open `myapp.cfg` and change the command-line template to the following:

```
<cmdlineapp>./bin/myapp.sh -f _ei_files -e _ei_outline </cmdlineapp>
```

As in the case of files, before executing the above command-line the server will replace `_ei_outline` by the list of received entities. Let us now change `myapp.sh` to process these entities in some way, e.g., printing them on the standard output. Open `myapp.sh` and add the following lines at the end:

```
1 echo ""
2 echo "Selected entities:"
3 echo ""
4 for e in $entities
5 do
6     echo "- $e"
7 done
```

This code simply print the entities in separated lines. Again, the external script `parse_params.sh` parses the command-line and stores the list of entities in the variable `entities`.

Go back to the web-client, reload the page, select some files and entities and execute the MFA application to see the result of the last changes. It is always recommended to check that the application works correctly from a shell first.

2.4 Passing Parameters to Your Application

In addition to input files and outline entities, real applications receive other parameters to control different aspects. In this section we explain how to declare parameters in the `EASYINTERFACE` framework such that (1) they automatically appear in the web-client (or any other client) so the user can set their values; and (2) the selected values are passed to the application when executed.

Let us start by modifying `myapp.sh` to accept some command-line parameters: we add a parameter “-s” to indicate if the received outline entities should be printed; and “-c W” that takes a value `W` to indicate what to count in each file — here `W` can be “lines”, “words” or “chars”. For example, `myapp.sh` could then be invoked as follows:

```
> myapp.sh -f file1.c file2.c file3.c -e sum.c:main sum.c:sum -s -c words
```

To support these parameters, change the content of `myapp.sh` to the following:

```
1 #!/bin/bash
2
3 . default/parse_params.sh
4
5 echo "I've received the following command-line parameters:"
6 echo ""
7 echo "  $@"
8
9 echo ""
10 echo "File statistics:"
11 echo ""
12
```

```

13 case $whattocount in
14     lines) wcparam="-l"
15     ;;
16     words) wcparam="-w"
17     ;;
18     chars) wcparam="-m"
19     ;;
20 esac
21
22 for f in $files
23 do
24     echo " - $f has " `wc $wcparam $f | awk '{print $1}` ` $whattocount
25 done
26
27 if [ $showoutline == 1 ]; then
28     echo ""
29     echo "Selected entities:"
30     echo ""
31     for e in $entities
32     do
33         echo "- $e"
34     done
35 fi

```

Compared to the previous script, you can notice that: we added lines 13-20 to take the value of “-c” into account when calling wc at line 24; and in lines 27-35 we wrapped the loop that prints the outline entities with a condition. Note that `parse_params.sh` sets the variable `whattocount` to the value of “-c”, and sets `showoutline` to 1 if “-s” is provided in the command-line and to 0 otherwise. Before proceeding to the next step, execute the script from a shell to make sure that it works correctly.

Our goal is to show these parameters in the web-client (or any other client), so the user can select the appropriate values before executing the application. The `EASYINTERFACE` framework provides an easy way to do this, all we have to do is to modify `myapp.cfg` to include a description of the supported parameters. Open `myapp.cfg` and add the following inside the `app` tag (e.g., immediately after `execinfo`):

```

<parameters prefix = "-" check="false">
  <selectone name="c">
    <desc>
      <short>What to count</short>
      <long>Select the elements you want to count in each input file</long>
    </desc>
    <option value="lines">
      <desc>
        <short>Lines</short>
        <long>Count lines</long>
      </desc>
    </option>
    <option value="words">
      <desc>
        <short>Words</short>
        <long>Count words</long>
      </desc>
    </option>
    <option value="chars" >
      <desc>
        <short>Chars</short>
        <long>Count characters</long>
      </desc>
    </option>
  </selectone>
  <default value="lines"/>

```

```

</selectone>
<flag name="s">
  <desc>
    <short>Show outline</short>
    <long>Show the selected outline entities</long>
  </desc>
  <default value="false"/>
</flag>
</parameters>

```

Let us explain the different elements of the above XML snippet. The tag `parameters` includes the definition of all parameters. The attribute `prefix` is used to specify the symbol to be attached to the parameter name when passed to the application, for example, if we declare a parameter with name "c" the server will pass it to the application as "-c". Note that this attribute can be overridden by each parameter. The attribute `check` tells the server to check the correctness of the parameters before passing them to the application, i.e., that they have valid values, etc. The tag `selectone` defines a parameter with `name` "c" that can take one value from a set of possible ones. For example, the web-client will view it as a ComboBox. The `desc` environment contains a text describing this parameter and is used by the client when viewing this parameter graphically. The `option` tags define the valid values for this parameter, from which one can be selected, and the `default` tag defines the default value. The `desc` environment of each `option` contains a text describing this option, e.g., the `short` description is used for the text in the corresponding ComboBox. The tag `flag` defines a parameter with name "s". This parameter has no value, it is either provided in the command-line or not, and its `default` value is "false", i.e., not provided. For the complete set of parameters supported in EASYINTERFACE see [PARAMETERS] in Chapter 3 in the EASYINTERFACE manual (<https://github.com/abstools/easyinterface>).

Go to the web-client, reload the page, and click on the `Settings` button and look for the tab with the title MFA. You will now see the parameters declared above in a graphical way where you can set their values as well. When you click on the `Apply` button, the web-client will pass these parameters to the server, however, we still have to tell the server how to pass these parameters to `myapp.sh`. Open `myapp.cfg` and change the `cmdlineapp` template to the following:

```

<cmdlineapp>./bin/myapp.sh -f _ei_files -e _ei_outline _ei_parameters </cmdlineapp>

```

As in the case of `_ei_files` and `_ei_outline`, the server will replace `_ei_parameters` by the list of received parameters before executing the command-line. Execute the MFA application from the web-client with different values for the parameters to see how the output changes.

2.5 Using the EASYINTERFACE Output Language in Your Application

In the example that we have developed so far, the web-client simply printed the output of `myapp.sh` in the console area. This is the default behavior of the web-client if the output does not follow the EASYINTERFACE Output Language (EIOL), which is a text-based language that allows generating more sophisticated output such as highlighting lines, adding markers, etc. In this section we will explain the basics of this language by extending `myapp.sh` to use it, for more details see Chapter 5 in the EASYINTERFACE manual.

An output in EIOL is an XML structure of the following form:

```

<eiout>
  <eicommands>
    [EICOMMAND]*
  </eicommands>
  <eiactions>
    [EIACTION]*
  </eiactions>
</eiout>

```

where (1) `eiout` is the outermost tag that includes all the output elements; (2) `[EICOMMAND]*` is a list of commands to be executed; and (3) `[EIACTION]*` is a list of actions to be declared. An `[EICOMMAND]` is an

instruction like: *print a text on the console, highlight lines 5-10, add marker at line 5*, etc. An [EIACTION] is an instructions like: *when the user clicks on line 13, highlight lines 20-25*, etc. In the rest of this section we discuss some commands and actions that are supported in EIOL, for the complete list see Chapter 5 in the EASYINTERFACE manual.

2.5.1 Printing in the Console Area

Recall that when the EIOL is used, the web-client does not redirect the output to the console area and thus we need a command to print in the console area. The following is an example of a command that prints "Hello World" in the console area:

```
<printonconsole consoleid="1" consoletitle="Welcome">
  <content format="text">
    Hello World
  </content>
</printonconsole>
```

The value of the `consoleid` attribute is the console identifier in which the given text should be printed (e.g., in the web-client the console area has several tabs, so the identifier refers to one of those tabs). If a console with such identifier does not exist yet, a new one, with a title as specified in `consoletitle`, is created. If `consoleid` is not given the output goes to the default console. Inside `printonconsole` we can have several `content` tags which include the content to be printed (in the above example we have only one). The attribute `format` indicates the format of the content. In the above example it is plain 'text', other formats are supported as well, e.g., 'html'.

Let us change `myapp.sh` to print the different parts of its output in several consoles. Open `myapp.sh` and change its content to the following:

```
1 #!/bin/bash
2
3 . default/parse_params.sh
4
5 echo "<eiout>"
6 echo "<eicommands>"
7 echo "<printonconsole>"
8 echo "<content format='text'>"
9 echo "I've received the following command-line parameters:"
10 echo ""
11 echo "  $@"
12 echo "</content>"
13 echo "</printonconsole>"
14
15 echo "<printonconsole consoleid='stats' consoletitle='Statistics'>"
16 echo "<content format='html'>"
17 echo "File statistics:"
18 echo "<div>"
19 echo "<ul>"
20
21 case $whattocount in
22   lines) wparam="-l"
23     ;;
24   words) wparam="-w"
25     ;;
26   chars) wparam="-m"
27     ;;
28 esac
29
30 for f in $files
31 do
```

```

32     echo " <li> $f has " `wc $wcpam $f | awk '{print $1}` $whattocount "</li>"
33 done
34 echo "</ul>"
35 echo "</div>"
36 echo "</content>"
37 echo "</printonconsole>"
38
39 if [ $showoutline == 1 ]; then
40     echo "<printonconsole consoleid='outline' consoletitle='Outline'>"
41     echo "<content format='html'>"
42     echo ""
43     echo "Selected entities:"
44     echo "<ul>"
45     echo ""
46     for e in $entities
47     do
48         echo "<li> $e </li>"
49     done
50     echo "</ul>"
51     echo "</content>"
52     echo "</printonconsole>"
53 fi
54 echo "</eicommands>"
55 echo "</eiout>"

```

The output of `myapp.sh` is given in EIOL, because at Line 5 we start the output with the tag `eiout` which we close at line 55. At Line 6 we start an `eicommands` tag, inside `eiout`, which we close at Line 54. Inside `eicommands` we have 3 `printonconsole` commands: the first one is generated by lines 7-13; the second by lines 15-37; and the last one by lines 40-52. Note that the first command uses the default console, while the last two use different consoles. Note also that the content in the last two is given in HTML. Execute `myapp.sh` in a shell first to check that it works correctly, and then execute the MFA application from the web-client to see the effect of these changes.

2.5.2 Adding Markers

Next we explain a command for adding a marker next to a code line in the editor area. The following is an example of such command:

```

<addmarker dest="path" outclass="info">
  <lines>
    <line from="4" />
  </lines>
  <content format='text'>
    text to associated to the marker
  </content>
</addmarker>

```

The attribute `dest` indicates the *full path* of the file in which the marker should be added (as received from the server). The attribute `outclass` indicates the nature of the marker, which can be 'info', 'error', or 'warning'. This value typically affects the type/color of the icon to be used for the marker. The tag `lines` includes the lines in which markers should be added, each line is given using the tag `line` where the `from` attribute is the line number (`line` can be used to define a region in other commands, this is why the attribute is called `from`). The text inside the `content` tag is associated to the marker (as a tooltip, a dialog box, etc., depending on the client).

Let us modify `myapp.sh` to add a marker at Line 1 of each file that it receives. Open `myapp.sh` and add the following code snippet immediately before 54 of the previous script (i.e., immediately before closing the `eicommands` tag):

```

1 for f in $files
2 do
3 echo "<addmarker dest='$f' outclass='info'>"
4 echo "<lines><line from='1' /></lines>"
5 echo "<content format='text'> text for info marker of $f</content>"
6 echo "</addmarker>"
7 done

```

Lines 3-6 generate the actual command to add a marker for each file passed to `myapp.sh`. Execute `myapp.sh` in a shell first to check that it works correctly, and then execute the MFA application from the web-client to see the effect of these changes.

2.5.3 Highlighting Code Lines

The following command can be used to highlight code lines:

```

<highlightlines dest="path" outclass="info" >
  <lines>
    <line from="5" to="10"/>
  </lines>
</highlightlines>

```

Attributes `dest` and `outclass` are as in the `addmarker` command. Each `line` tag defines a region to be highlighted. E.g., in the above example it highlights lines 5-10. You can also use the attributes `fromch` and `toch` to indicate the columns in which the highlight starts and ends respectively.

Let us modify `myapp.sh` to highlight lines 5-10 of each file that it receives. Open `myapp.sh` and add the following code snippet immediately before the instruction that closes the `ecommands` tag:

```

1 for f in $files
2 do
3 echo "<highlightlines dest='$f' outclass='info'>"
4 echo "<lines><line from='5' to='10' /></lines>"
5 echo "</highlightlines>"
6 done

```

Execute `myapp.sh` in a shell first to check that it works correctly, and then execute the MFA application from the web-client to see the effect of these changes.

2.5.4 Adding Inline Markers

Inline markers are widgets placed inside the code. They typically include some read-only text. The following command adds an inline marker:

```

<addinlinemarker dest="path" outclass="info">
  <lines><line from="15" /></lines>
  <content format="text">
    Text to be viewed in the inline marker
  </content>
</addinlinemarker>

```

Attributes `dest` and `outclass` are as in the `addmarker` command. Each `line` tag defines a line in which a widget, showing the text inside the `content`, is added. Note that some clients, e.g., the web-client, allow only plain 'text' content.

Let us modify `myapp.sh` to add an inline marker at line 15 of each file that it receives. Open `myapp.sh` and add the following code snippet immediately before the instruction that closes the `ecommands` tag:

```

1 for f in $files
2 do
3 echo "<addinlinemarker dest='$f' outclass='info'>"
4 echo "  <lines><line from='15' /></lines>"
5 echo "  <content format='text'> Awesome line of code!!</content>"
6 echo "</addinlinemarker>"
7 done

```

Execute `myapp.sh` in a shell first to check that it works correctly, and then execute the MFA application from the web-client to see the effect of these changes.

2.6 Opening a Dialog Box

The following command can be used to open a dialog box with some content:

```

<dialogbox outclass="info" boxtitle="Done!" boxwidth="100" boxheight="100">
  <content format="html">
    text to be shown in the dialog box
  </content>
</dialogbox>

```

The dialog box will be titled as specified in `boxtitle`, and it will include the content as specified in the `content` environments. The attributes `boxwidth` and `boxheight` are optional, they determine the initial size of the window.

Let us modify `myapp.sh` to open a dialog box with some message. Open `myapp.sh` and add the following code snippet immediately before the instruction that closes the `ecommands` tag:

```

1 echo "<dialogbox outclass='info' boxtitle='Done!' boxwidth='300' boxheight='100'>"
2 echo "  <content format='html'>"
3 echo "    The <span style='color: red'>MFA</span> analysis has been applied."
4 echo "    You can see the output in the result in the text area and the corresponding"
5 echo "    program files."
6 echo "  </content>"
7 echo "</dialogbox>"

```

Execute `myapp.sh` in a shell first to check that it works correctly, and then execute the MFA application from the web-client to see the effect of these changes.

2.7 Adding Code Line Actions

A *code line action* defines a list of commands to be executed when the user clicks on a line of code (more precisely, on a marker placed next to the line). The commands can be any of those seen above. The following is an example of such action:

```

<oncodelineclick dest="/Examples_1/iterative/sum.c" outclass="info" >
  <lines><line from="17" /></lines>
  <ecommands>
    <highlightlines>
      <lines>
        <line from="17" to="19"/>
      </lines>
    </highlightlines>
    <dialogbox boxtitle="Hey!">
      <content format="html">
        Click on the marker again to close this window
      </content>
    </dialogbox>
  </ecommands>
</oncodelineclick>

```

```
</eicommands>  
</oncodelineclick>
```

First note that the above XML should be placed inside the `eiactions` tag (that we have ignored so far). When the above action is executed, by the web-client for example, a marker (typically an arrow) will be shown next to line 17 of the file `/Examples_1/iterative/sum.c`. Then, if the user clicks on this marker the commands inside the `eicommands` tag will be executed, and if the user clicks again the effect of these commands is undone. In the above case a click highlights lines 17-19 and opens a dialog box, and another click removes the highlights and closes the dialog box. Note that the commands inside `eicommands` inherit the `dest` and `outclass` attributes of `oncodelineclick`, but one can override them, e.g., if we add `dest="/Examples_1/iterative/fact.c"` to the `highlightlines` command then a click highlights lines 17-19 of `fact.c` instead of `sum.c`.

Let us modify `myapp.sh` to add a code line action, as the one above, for each file that it receives. Open `myapp.sh` and add the following code snippet immediately before the instruction that closes the `eiout` tag (i.e., after closing `eicommands`):

```
1 echo "<eiactions>"  
2  
3 for f in $files  
4 do  
5   echo "<oncodelineclick dest='$f' outclass='info' >"  
6   echo "<lines><line from='17' /></lines>"  
7   echo "<eicommands>"  
8   echo "<highlightlines>"  
9   echo "<lines><line from='17' to='19' /></lines>"  
10  echo "</highlightlines>"  
11  echo "<dialogbox boxtitle='Hey!'> "  
12  echo "<content format='html'>"  
13  echo "Click on the marker again to close this window"  
14  echo "</content>"  
15  echo "</dialogbox>"  
16  echo "</eicommands>"  
17  echo "</oncodelineclick>"  
18 done  
19  
20 echo "</eiactions>"
```

Note that at line 1 we open the tag `eiactions` and at line 20 we close it. The rest of the code simply prints a code line action as the one above for each file. Execute `myapp.sh` in a shell first to check that it works correctly, and then execute the MFA application from the web-client to see the effect of these changes.

2.8 Adding OnClick Actions

OnClick actions are similar to code line actions. The difference is that instead of assigning the action to a line of code, we can assign it to any HTML tag that we have generated. For example, suppose that at some point the application has generated the following content in the console area:

```
<content format="html" />  
  <span style="color: red;" id="err1">10 errors</span> were found in the file sum.c  
</content>
```

Note that the text "10 errors" is wrapped by a `span` tag with an identifier `err1`. The OnClick action can assign a list of commands to be executed when this text is clicked as follows:

```
<onclick>  
  <elements>  
    <selector value="#err1" />
```



```

</elements>
<eicommands>
  <dialogbox boxtitle="Errors">
    <content format="html">
      There are some variables used but not declated
    </content>
  </dialogbox>
</eicommands>
</onclick>

```

It is easy to see that this action is very similar to `oncodelineclick`, the difference is that instead of `lines` we now use `elements` to identify those HTML elements a click on which should execute the commands.

Let us modify `myapp.sh` to add an OnClick action assigned to the list of files that it prints on the console. First look for the first occurrence of

```
1 echo "<ul>"
```

which should be at line 19, and replace it by

```
1 echo "<ul style='background: yellow;' id='files'>"
```

This change will give the list of files that we print in the console (i.e., the corresponding HTML) the identifier `files`, and will change its background color to yellow. Next add the following code immediately before the instruction that closes `eiactions`:

```

1 echo "<onclick>"
2 echo "<elements>"
3 echo "<selector value='#files' />"
4 echo "</elements>"
5 echo "<eicommands>"
6 echo "<dialogbox boxtitle='Errors'> "
7 echo "<content format='html'>"
8 echo "There are some variables used but not declated"
9 echo "</content>"
10 echo "</dialogbox>"
11 echo "</eicommands>"
12 echo "</onclick>"

```

This defines an OnClick actions such that when clicking on the list of files in the console area (anywhere in the yellow region) should open a dialog box. Execute `myapp.sh` in a shell first to check that it works correctly, and then execute the MFA application from the web-client to see the effect of these changes.

3 The Envisage Setting

TODO: samir will add

4 Further Reading

TODO: samir will add