

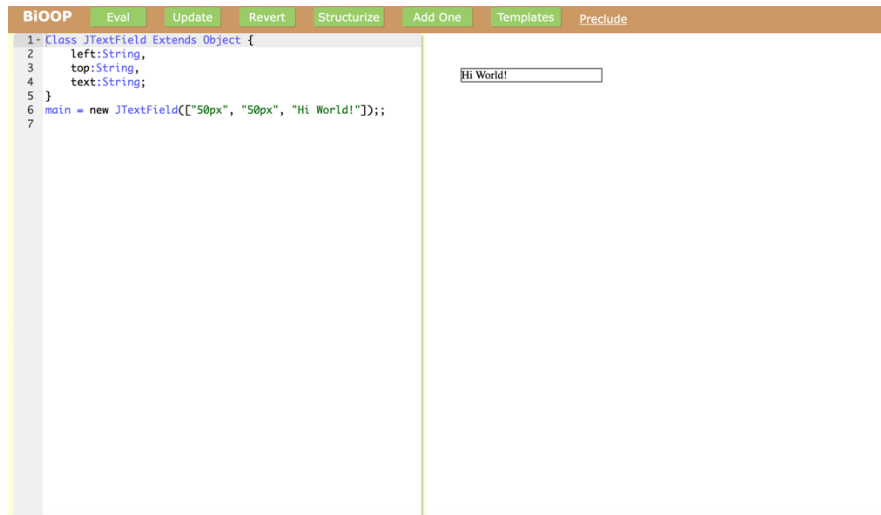
Overview of BiOOP

Content

INTRODUCTION	1
GETTING STARTED	2
<i>Where it can be obtained</i>	2
<i>Setup</i>	2
<i>Basic Testing</i>	4
STEP-BY-STEP	5
<i>How to reproduce the results presented in the paper</i>	5
<i>User Guide of BiOOP</i>	8

Introduction

Our paper presents a bidirectional object-oriented language BiFJ (a Featherweight-Java-like language) that supports programmatic and direct manipulation of objects. The artifact BiOOP is a programming environment to support BiFJ in single web-page GUI design. As shown below, BiOOP supports developers not only to write object-oriented programs in the left editor and get the output (e.g., a web page) in the right Output window, but also to directly manipulate the output on the right, and automatically synchronize the manipulated output with the object-oriented program.



In Section 5 of our paper, we demonstrate the expressiveness of our language (BiFJ) and efficiency of our tool (BiOOP) in the context of object-oriented GUI programming. There are two main claims in our paper.

1. BiOOP can be used to develop typical GUI programs. In Section 5.1, we collect 11 benchmark programs of Java Swing, a classical GUI framework in Java, and reproduce

them using BiFJ in BiOOP.

2. BiOOP can efficiently response to the developers' edits by propagating the edits bidirectionally within a reasonable time cost. We instrument a timer mechanism in our implementation and measure the execution time for the 11 benchmark programs.

We're applying for the "Reusable" badge. We believe that the artifact deserves that badge because BiOOP can support the main claims of the paper and the code structure is clear and reusable. In addition to placing it into a public accessible archival repository, we build a docker image. The docker image is already configured with Elm v0.19 and contains the source code (including the 11 benchmark programs) and compiled files, so developers can use it directly or do extended research. The code structure is as follows.

- Main implementation: `./src/OOP`
- Compiled file: `./oop-index.html`
- Compiled file with experiment settings: `./oop-index-text.html`
- Benchmark programs: `./src/OOP/Examples/OOPSLA_23`
- Experimental result: `./src/OOP/Examples/OOPSLA_23/Result.xlsx`

Getting Started

Where it can be obtained

1. Docker Image (public accessible):

https://drive.google.com/file/d/1IGXwWXOOnOaWEtQnIGJAbOB3Ju0a1Xom/view?usp=share_link

Or, directly download it from the Submission website

2. Open Source Library:

<https://github.com/xingzhang-pku/BiOOP>

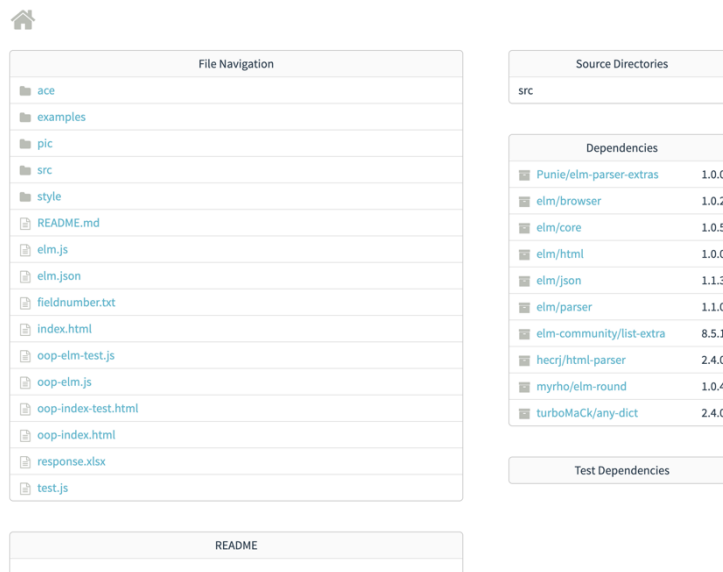
Setup

There are two ways to install our artifact: one way is to configure the environment directly on localhost; the other way is to use docker.

Install using Docker (recommended)

1. Download the docker image file (BiOOP.tar)
2. Import the image and name it bioop
 - `docker import BiOOP.tar bioop:latest`
3. Check if import successfully
 - `docker images`
4. Create a container with port mapping and name it BiOOP. (Port 8000 of the container is mapped to port 8080 of the host.)

- `docker run -ti -d --name BiOOP -p 8080:8000 -it bioop /bin/bash`
5. Enter project
 - `cd /root/BiOOP`
 6. Compile project
 - `elm make src/OOP/Main.elm --output=oop-elm.js`
 7. Starts a server (at <http://localhost:8000>)
 - `elm reactor`
 8. In the host, visit url: <http://localhost:8080>
- You will see the source code of BiOOP in the docker container as shown below.



9. Click oop-index.html in the web page you just opened.
- Steps 9 and 10 can also be achieved by directly accessing <http://localhost:8080/oop-index.html>.



Install Locally

1. Install Elm v0.19 (<https://guide.elm-lang.org/install/elm.html>).
2. Download Source Code (<https://github.com/xingzhang-pku/BiOOP>).
3. Enter BiOOP folder (The root directory containing the src folder).
4. Compile Project.
 - `elm make src/OOP/Main.elm --output=oop-elm.js`
5. Open oop-index.html using browser.

Browser Running oop-index.html (Important!)

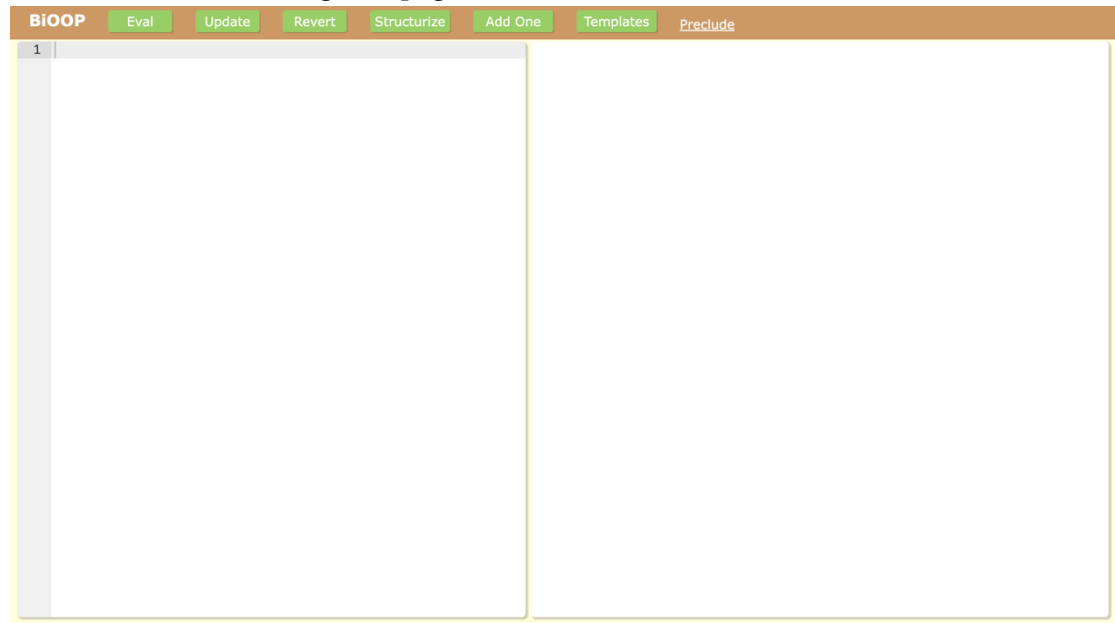
The **Chrome** browser is recommended. If your Chrome has a lot of extensions, please test it in the **incognito mode**. The guide to open incognito mode is as follows or visit <https://support.google.com/chrome/answer/95464?hl=en&co=GENIE.Platform=Desktop> for more details. The reason why you need to turn on the incognito mode is because a

cross-origin request issue and it is caused by various Chrome Extensions.

1. On your computer, open Chrome.
2. At the top right, click More  > **New Incognito Window**.
3. A new window appears. In the top corner, check for the Incognito icon .

A Method to Test the Installation

You will see the following web page.



Basic Testing

We use this small example to show: our tool allows developers to modify the output, thereby modifying the corresponding object-oriented program, so that the program can run again to get the modified output.

1. Write an initial object-oriented program in the left editor of our tool.

```
Class JTextField Extends Object {  
    left:String,  
    top:String,  
    text:String;  
}  
main = new JTextField(["50px", "50px", "Hello World!"]);;
```

2. Click the “Eval” button. You will see the output of the above program in the Output window like the below.



3. Change “Hello World!” to “Hi World!” and click the “Update” button. Then Line 6 of the program will update “Hello World!” to “Hi World!”.



4. Click the “Eval” button to run the program again. You will see the output is just the same as the modified one in Step 3.

Step-by-Step

How to reproduce the results presented in the paper

To demonstrate that BiOOP can be used to develop typical GUI programs, we reproduced 11 benchmark programs of Java Swing using BiFJ in BiOOP. To demonstrate that our tool BiOOP can efficiently response to the developers’ edits by propagating the edits bidirectionally within a reasonable time cost, we instrument a timer mechanism in our implementation and measure the execution time for the 11 benchmark programs.

For reproducing 11 benchmark programs

There are 11 benchmark examples in the paper. Their overall information and screenshots are shown in the following. For more details, see Section 5.1 in our paper.

Table 3. Benchmarks

ID	Example	LOC	#Cls	Depth of CT	Width of CT	#Obj
1	File Explorer	188	9	4	4	21
2	Source Getter	189	9	4	3	5
3	IP Finder	201	9	4	3	5
4	Number Puzzle	222	7	4	2	12
5	Picture Puzzle	238	7	4	2	13
6	Online Exam	245	10	5	2	54
7	Word Counter	274	12	4	3	9
8	Tic Tac Toe	287	8	4	2	13
9	NotePad	331	11	4	3	37
10	Calculator	371	11	5	3	29
11	Data Sync	631	22	5	10	97

LOC: Lines of code in BiFJ; **#Cls**: Numbers of classes; **Depth of CT**: Length of the longest inheritance chain in the class table; **Width of CT**: Maximum number of sub-classes of the same class; **#Obj**: Numbers of objects;

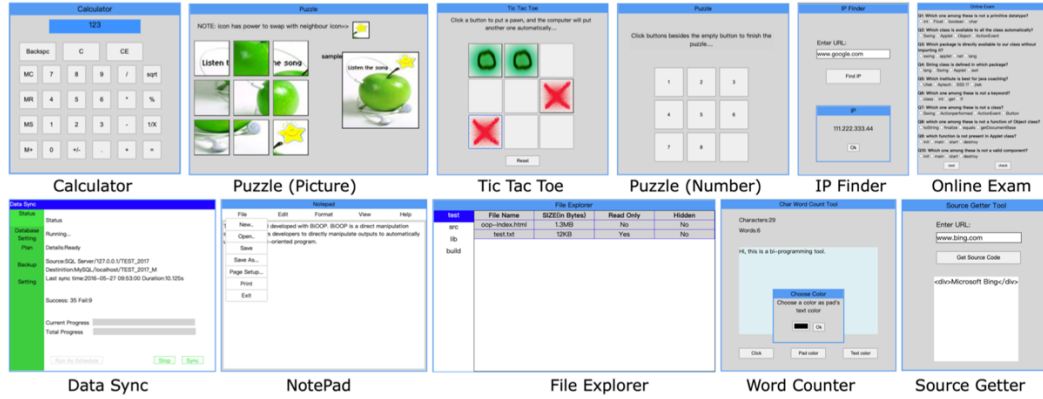


Fig. 17. Screenshots of Applications

The initial code of benchmark examples can be found through visiting <http://localhost:8080/src/OOP/Examples/OOPSLA%2023/> (i.e., Calculator, DataSync, FileExplorer, IPFinder, NotePad, NumPuzzle, OnlineTest, PicPuzzle, SourceGetter, Tic Tac Toe, WordCounter). Or you can download the source code to find path “./src/OOP/Examples/OOPSLA 23”.

Each example file contains two parts: a **program** (ending with `main` definition) and some **templates** separated by **dashed lines** as shown below. (To simplify, **templates can be ignored** because they are already built into the file “src/OOP/Objects/Templates.elm”).

```

main = (backspcBtn->setHint) "Backspc";
      (cBtn->setHint) "C";
      (calculator->setMembers) btns;
      (calculator->setTitle) "Calculator";
      calculator;;

-----

("JButton"
, "new JButton([left, top, height, width, hint, id, onclick])"
, "" "Html.div
  [ ["margin-left", left]
  , ["margin-right", "auto"]
  , ["margin-top", top]
  , ["margin-bottom", "auto"]
  , ["position", "absolute"]
  ]
  [ Html.button [{"height", height},
                  {"width", width}]
                  [{"id", id},
                  {"onclick", onclick}]
                  [hint]
  ]
),

```

Program

Templates

To reproduce each example, you can

- Copy the program (the part of the file above the dashed line) into the code editor, then click the "Eval" button to run, and get the output in the right Output window.
- Modify the output (see the following section "User Guide of BiOOP" for modifications) and click the "Update" button to get the automatically updated program.
- Click the "Eval" button again to see if the updated program runs to get the modified output.

For reproducing the experiments

We instrument a timer mechanism in the compiled oop-elm.js file and measure the execution time for the 11 benchmark programs. For each program, we perform 9-11 direct manipulations on its execution output recorded in the following table.

Example	Direct Manipulation				
Calculator	modify the color of displaylabel adjust the backspace button's size rewrite calculator's title rewrite calculator's title rewrite backspace and C's hint rewrite hint of backspace button add a digitbtn delete Backspace button change a DisplayLabel to a JLabel	Number Puzzle	rewrite puzzle's title rewrite the text of label rewrite the hint of button delete label delete a button add a puzzle button set the size of puzzle frame swap the hint of two buttons by click	IP Finder	add a JLabel add a button add a JTextField delete TextField rewrite url rewrite hint rewrite IP Finder's title rewrite IP Finder's title rewrite hint of findip button set size of findip button
File Explorer	add a File delete a MyTid delete a Folder change a MyTh to a MyTid rename oop-index.html set current folder's color set the width of file explorer rewrite a Folder's name rewrite file explorer's title rewrite file explorer's title	Tic Tac Toe	add a label add a tictactoe button rewrite title rewrite the text of label rewrite the hint of reset button set the label's size set the label's color delete reset button	Notepad	add a JMenu add a JMenuItem delete a JMenu delete a JMenuItem rewrite notepad's title rewrite filemenu's text rewrite text of default file rewrite filemenu and editmenu's text change a JMenu to a JMenuItem change a JMenuItem to a JMenuItem set the background color of editor
		Source Getter	rewrite url rewrite title rewrite url's sourcecode add a JLabel add a TextField delete label delete TextField set label's color set label's size	Online Exam	add a question add an option for Q1 delete Q2 delete an option of Q1 rewrite text of Q1 rewrite an option of Q1 rewrite onlinetest's title rewrite hint of next button set the size of next button
		Word Counter	delete a button add a colorchooser add a click button change text of label change mylabel to jlabel change the text to be counted rewrite title rewrite the hint of 3 buttons set the color of label set the size of word counter	Picture Puzzle	add a dialog add an Image add a puzzle button delete hintimg rewrite picpuzzle's title rewrite the hint of hintimg rewrite picpuzzle's title rewrite the hint of sampleimg change the length of hintimg's hint
		Data Sync	rewrite title rewrite the hint of plan rewrite the hint of stop button rewrite the text of status label rewrite the hint of stop and sync button add a progressbar delete stop button change a NActiveBtn to ActiveBtn set the color of status label set the margin of stop button		

The Performance Experiment Results is recorded in the following table.

Table 4. Performance Experiment Results

ID	#DMP	Fwd	F:Tot	Fastest			Slowest			Avg		
				Bwd	CTL	B:Tot	Bwd	CTL	B:Tot	Bwd	CTL	B:Tot
1	10	1	655	10	1	626	11	1	799	11	1	671±26
2	9	1	143	5	1	114	5	1	161	5	0	145±29
3	10	1	133	5	1	118	5	1	165	5	1	151±13
4	10	3	313	29	1	357	255	1	581	38	1	406±73
5	9	3	378	34	1	447	152	1	630	32	1	495±14
6	10	3	1394	38	1	1540	255	1	1946	197	1	1698±152
7	11	3	329	20	1	329	19	2	397	21	1	377±26
8	10	6	354	2017	2	2642	2030	2	2706	2013	2	2684±31
9	11	2	1008	12	0	912	12	1	1242	11	1	1124±117
10	9	3	1015	26	2	1147	677	4	1891	26	2	1189±22
11	10	2	3855	499	1	4533	514	2	4802	502	2	4679±162

#DMP: Numbers of Direct Manipulations; Fwd: Time to forward evaluation in milliseconds for initial programs; F:Tot: Total time between clicking the "Eval" button and getting the output; Bwd: Time to backward evaluation; CTL: Time to class table lifting; B:Tot: Total time between clicking the "Update" button and getting the updated program. The data 0 in the table is due to less than 0.5 before retaining integers.

To simplify, we add timing statements (like the following) to the "oop-elm.js" file and save it as the "oop-elm-test.js" file. We then create a new HTML file "oop-index-test.html" by copying "oop-index.html" and add "oop-elm-test.js" file to it. Therefore, reviewers can **directly run the "/oop-index-test.html" to measure the execution time**. For more details, see Section 5.2 in our paper.

```

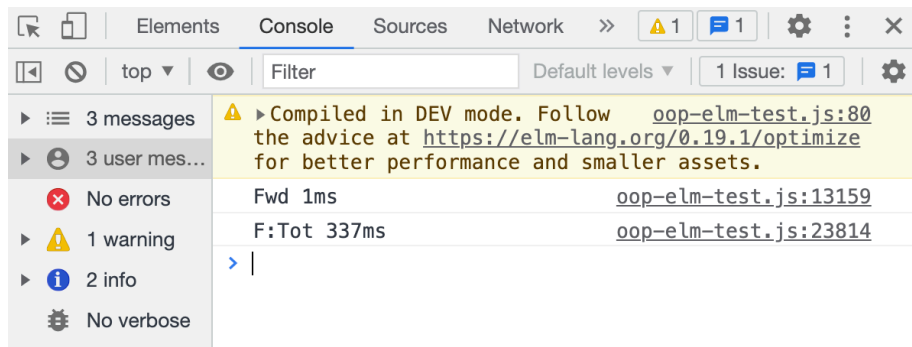
13156   var evalStart = new Date().getTime();
13157   var _v2 = A4($author$project$00P$Eval$eval, _List_Nil, _List_Nil, withGui, pTerm);
13158   var evalEnd = new Date().getTime();
13159   console.log("Fwd " + (evalEnd - evalStart) + "ms");

```

To test each benchmark example, you can

- Copy the program into the code editor, then click the "Eval" button to run, and get the

output in the right Output window. Then the running time “Fwd” and “F:Tot” will show in the console of the Develop Tool like the following.



- b) Modify the output and click the “Update” button. Then the running time “Bwd”, “CTL”, and “B:Tot” will show like the following.



User Guide of BiOOP

Grammar

To make it easier for users to learn BiFJ, we give some syntax examples below.

1. Class Declarations

```
Class JFrame Extends Container {
    title:Ref<String>,
    members:Ref<List<JComponent>>;

    setMembers(members){
        this.members:=members
    };
    setTitle(title){
        this.title:=title ++ ">"
    };
}
```

2. Data Structures

- a) String: "abc"
- b) Tuple: (1,2)
- c) List: [1,2,3] or 1::2::[]

3. Arithmetic Expressions

- a) Binary Operators: `* / // + - ++ < > <= >= == && ||`
- b) Unary Operators: `- ~`
- 4. Sequence: `s1;s2`
- 5. Conditions
 - a) If Statement: `if t1 then t2 else t3`
 - b) Case Statement:


```
case ls of
  [] => []
  | x :: xs => xs
```
- 6. References
 - a) Reference Creations: `ref 3`
 - b) Dereferences: `!r`
 - c) Assignments: `t1:=t2`
- 7. Statements: `var = term ;;`
- 8. Objects
 - a) Field Access: `t.f`
 - b) Method Invocation: `(object->method) args`
 - c) Object Creations: `new Class([args])`
- 9. Example


```
Class JComponent Extends Object {}
Class JMenuItem Extends JComponent {
  title:Ref<String>;
  setTitle(title){
    this.title:=title++"!!!"
  };
}
Class JMenu Extends JMenuItem {
  id:String,
  items:List<JMenuItem>;
}
item1 = new JMenuItem([ref "Item1"]);;
item2 = new JMenuItem([ref "Item2"]);;
menu = new JMenu([ref "Menu", "menu-id", [item1, item2]]);;
main = (item1->setTitle) "Menu-Item";
      (menu->setTitle) "Test Menu";
menu;;
```

Direct Manipulations of Output

Our tool support developers not only to manipulate the values (e.g., strings) in the HTML page directly or use the DevTools that come with the browser, but also manipulate the HTML page's GUI components by modifying the program's object structure. Value manipulation is straightforward, therefore we focus on structure manipulations. For more details, see Section 2.2 in our paper.

Modifying Types

For example, if we want to modify the type of the “Save” menu to a single menu item. First, click the “Structurize” button on the top menu bar of BiOOP. Then, right-click the “Save” menu and a small menu pops up with three items (i.e., Type: Menu, Modify Type, Delete). Right-click the “Modify Type” button and a class list (i.e., MenuItem, Object) will appear. Left-click the “MenuItem” and the modification is completed.

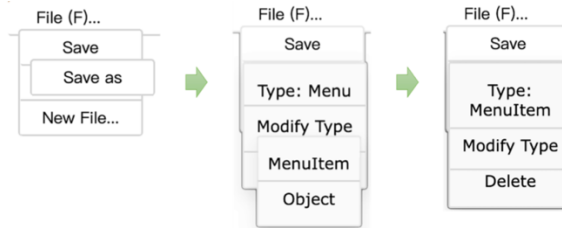


Fig. 7. Modifying Component Types: modifying the “Save” menu to the “MenuItem” type

Deleting Components

Deleting components is similar to modifying types. The previous steps are the same, except that you left-click the "Delete" button like the following example.

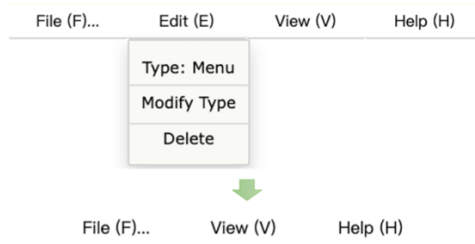


Fig. 8. Deleting Objects: deleting the “Edit (E)” menu on the top menu bar

Add Components

First, click the “Add One” button on the top menu bar of BiOOP and the ‘+’ symbols appear. Just click on the ‘+’ symbol where you want to add a component. Our tool copies the previous component by default to provide valid initial parameters. You can modify the text of the default added component to the desired one.

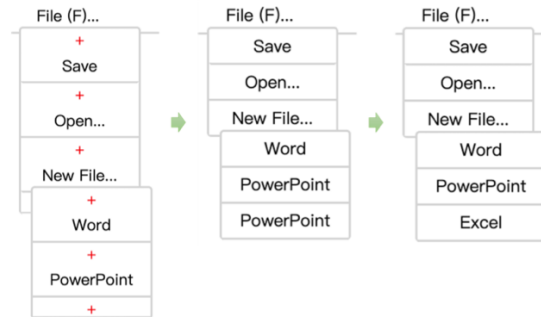


Fig. 9. Adding Objects: adding the “Excel” menu item to the “New File” menu