

DEVELOPER'S GUIDE How to develop plug-ins for Justinmind



Ν	ΗX I

Overview	3
Installing the SDK	4
First example: Print screen names	5
Configuring the Project	5
Building your first integration	8
Methods and Classes	10
Introduction	10
API library description	11
Main classes diagram	12
Components classes diagram	13
Classes referring to events	15
Second example: Translator	16
Extend Prototyper: Plug-ins	18
References	21
Appendix: Debug your plug-ins code	22





WHAT IS THIS DOCUMENT ABOUT?

Welcome to the developer documentation for Justinmind tools. If you want to write an add-on for Justinmind, you have come to the right place. Your add-on may be a plug-in or an entirely independent application that uses our APIs. This document covers both aspects: how to make a plug-in for Justinmind and how to build an entirely independent application that is able to read the contents in a .vp file (files made with Justinmind that contain all the information of a prototype).

You can use our APIs to reuse the work you did building your prototypes and generate code. In this document you will find a description of the classes included in our APIs and some examples that will help you to get started. You can find additional documents in <u>our web site</u>.





INSTALLING THE SOFTWARE DEVELOPMENT KIT (SDK)

The first thing you have to do if you want to code an integration with Justinmind is to <u>download our SDK</u>. This SDK is an Eclipse IDE with all the required libraries already included. Once downloaded just unzip it in a folder and launch it. It will request you to set another folder as 'workspace' for all your project files. Choose one and press ok. Then, a screen similar to this one will appear:



This is the typical welcome window for the Eclipse IDE. Just close it and let's build our first integration with Justinmind. Are you excited? I am!



FIRST EXAMPLE: PRINT SCREEN NAMES

FIRST EXAMPLE: PRINT SCREEN NAMES

You are all set and ready to build your first integration with Prototyper. In this case we are going to build a very simple code that reads a .vp file and prints by console the names of all the screens in that prototype. Let's start.

CONFIGURING THE PROJECT

Select File \rightarrow New \rightarrow Other... Then choose 'Plug-in Project (*IMPORTANT* **NOTE:** all the integrations must be coded in Java, if you don't know how to code in Java, well, you've just found a reason to learn it!. There is plenty of resources, tutorials and stuff to learn how to code in Java. Just google it, is not that hard)

New	_ D X
Select a wizard Create a Plug-in Project	
Wizards: type filter text	
 CVS Java Plug-in Development Category Definition Component Definition Extension Point Schema Feature Patch Feature Project Fragment Project Plug-in from Existing JAR Archives Plug-in Project Product Configuration Target Definition Vupdate Site Project Wate Assistance XML Examples 	
Sack Next > Finish	Cancel



FIRST EXAMPLE: PRINT SCREEN NAMES

Press 'Next' and then choose a name for your project. Press 'Next' again and check the options 'Generate an activator...' and 'This plug-in will make contributions to the UI'

New Plug-in Project						
Content Enter the data required to	generate the plug-in.					
Properties						
ID:	PrintScreenNames					
Version:	1.0.0.qualifier					
Name:	PrintScreenNames					
Provider:						
Execution Environment:	JavaSE-1.6					
Options Generate an activator, a Java class that controls the plug-in's life cycle Activator: printscreennames.Activator This plug-in will make contributions to the UI Enable API Analysis Rich Client Application						
Would you like to create	a rich client application? O Yes (a) No < Back Next > Finish Cancel					





Finally push 'Finish'. Your Eclipse IDE will look like this:

Plug-in Development - PrintScreenNames/I				
Eile Edit Source Refactor Navigate Se				
11-1100 \$+0-Q-	₩ @ • @ @ / • 1 1 • 1 • 1 • 1	\$ • \$ •		😭 🎝 Java 🔩 Plug-in "
📲 Package Ex 😂 🍂 Plug-ins 🐡 🗖	PrintScreenNames 23		- 0	🗄 Out 🖾 🔳 Tas 🔍 🗖
	Overview		0 · \$ · \$ 0 🔒	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
PendScreenName MarketsLaw MarketsLaw MarketsLaw MarketsLaw MarketsLaw MarketsLaw MANRESLAW MANRESLAW MANRESLAW	General Informatio This section devolves general information about this plug-in. D: Prefix-cerenNames Version 10.00 guidiffe Name: Prefix-cerenNames Product: Product:		Pag-in-Content The content of the play-in in marks up of two sections: If an excitation section is the play-in excitation is the play-in excitation is the excitation of the play-in excitation is the excitation of the excita	Overview Dependencies Dependen
	Update the classpath settings		To package and export the plug-in:	
	Overview Dependencies Runtime Extensions Exte	ension Points Buil	Id MANIFEST.MF build.properties	* *
	🖲 Error Log 🧟 Tasks 🔣 Problems 📮 Console	23		d ∎ • t3 • ° °)
	No consoles to display at this time.			
D ⁰ src - PrintScreenNames	J			

Now select the tab called 'Dependencies'. Push the 'add' button and write 'com.justinmind'. Select the three plug-ins listed and press 'ok'. Then select 'File \rightarrow Save'. Now you have a project configured to access to all the methods in the API.

You will need to follow the same steps every time you want to build a new integration for Justinmind.





BUILDING YOUR FIRST INTEGRATION

Ok, so you are ready now to build your first integration. This will be a very simple integration but you will see it will be easy to extend it to more complex cases. In this case we are going to create a little java application that reads all the screens inside a .vp file and print their names by console. Let's do it!

First create a class inside your source folder. Right click on the folder called 'src' and select 'New \rightarrow class'. Give it a name and check the option 'public static void main(String[] args)' like this:

lava Class	efault package is discouraged.	C
Source folger:	Example/src	Brgwse
Pac <u>k</u> age:	(default)	Bro <u>w</u> se
Enclosing type:		Bro <u>w</u> se
Na <u>m</u> e:	PrintScreenNames	
Modifiers:	gublic Odefault Oprivate Oprotected abstract Infinal Istatic	
Superclass:	java.lang.Object	Browsg
Interfaces:		Add
	(Remove
	bs would you like to create? @jeublic static yold main(String[] args) @ Constructors from superclass @ Ingerited abstract methods d comments? (Configure templates and default value <u>here</u>)	

Press the finish button. Then add the actual code for the integration until it looks like this:

	0.4. # 0. 004. 9	ALL ALL	1				E 3
SPI PD	() "PrintScreenNames.java ()	Ste. 10 181 181	V. 4 4. 4.			20	E Outline II III Task List
All of the second	<pre>pupper gen_ic_TIME import gen_ic_TIME import gen_ic_TIME import gen_ic_TIME import gen_ic_TIME import gen_ic_TIME import mo_putEntime import impor</pre>	heal) -cal JFG -cal JFG	nty(")svs.io.tmpd omy/File,separate sloaff().loaffoot Liowense(); mme());	otype(temp_director; re*unit*);	veftla, segaratore*next*);		<pre>prove the termination of termination of the termination of term</pre>
Commons-s	😢 Error Log 🕢 Tasks 🔝 Problems 🔅 🔲 Cor	nsole				1 × × 0	
freemarker-	0 kems Description	Resource Path	Lecation	Tune			
🚔 jærpc.jør - 🕻							
🚔 jdomjar - C							
jdomjar - C							
🚔 jdomjar - C							



FIRST EXAMPLE: PRINT SCREEN NAMES

Save your changes. Then right click on the class file you just created and select 'Run as \rightarrow Java Application'. The example you just coded will be executed allowing you to select a .vp file and see all its screens listed in the console.

Now let's take a closer look at the code. The first two lines use a file dialog to ask the user for a .vp file. But let's focus on the three lines after those lines of code:

String temp_directory = System.getProperty("java.io.tmpdir");
FileUtils.unZip(path_to_vp, temp_directory+File.separator+"work");
IPrototype prototype =
API.getPrototypeLoader().loadPrototype(temp_directory+File.separat
or+"work");

These are the lines of code that do all the magic. This piece of code will load all the prototype's information on the selected .vp file into the variable called 'prototype'. Using that variable and the methods in the API you can browse all the information in your prototypes and generate whatever you want with it. In this example we just read all the screens, templates and masters and we print their names by console.

In order to load all the information of a .vp file you need to extract the contents of the file into a folder. That's what the first two lines of this example do. Just remember to clean all those files when you are done with it using this line of code:

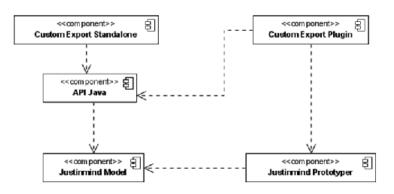
FileUtils.deepDeleteDirExceptRoot(temp_directory+File.separator+"w
ork");



METHODS AND CLASSES

METHODS AND CLASSES

Now let's take a look at the components this API is composed by.



As we have seen in the previous section the API is made of three plug-ins or jars (Java libraries). One is a library of some utilities to handle files and minor stuff. The other two are the ones that hold all the code to load a prototype.

The model are the actual classes that hold the information of the prototype. Those classes are highly complex, that's why another library has been made an placed over it. The API library is the one you must use when you code an integration with Justinmind. The direct access to the model library is highly discouraged because the classes there can change from one update of Justinmind to another. All the documents provided for the developers of plug-ins refer to the classes in the API library, not the ones in the model library.





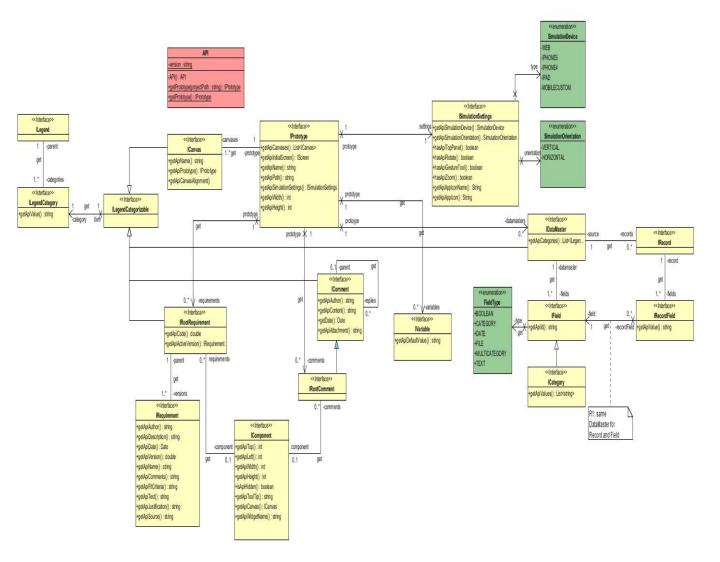
API LIBRARY DESCRIPTION

The main entry point to the API library is the class named 'API'. That class has static methods to load a prototype's file. There are two ways to load a prototype, one is using the path to the location of a .vp file. The other one is to load the 'current loaded prototype' and only applies to plug-ins that will be run from within Justinmind.

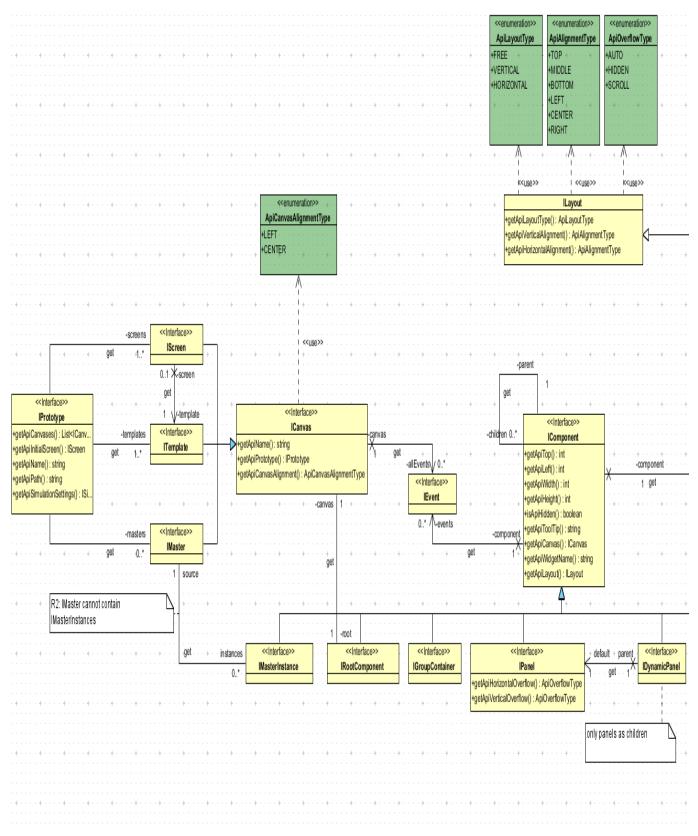
API
-version : string
+API(): API
+getPrototype(projectPath:string):IPrototype
+getPrototype(): IPrototype

All the classes in the api library are Java Interfaces and they are not intended to be subclassed or implemented by your own classes.

The rest of the classes in the library can be categorized in three groups: the main prototype classes, the classes for the components and the classes for the events. In the next pages you can find the UML diagrams for each of this groups (check the Javadoc documents for a detailed description of each class and method).

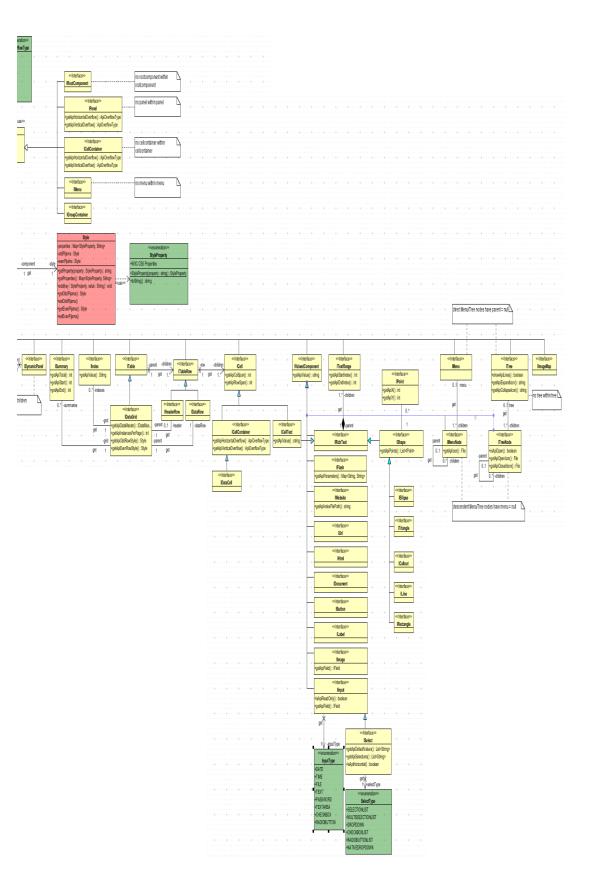


COMPONENTS CLASSES DIAGRAM (1)

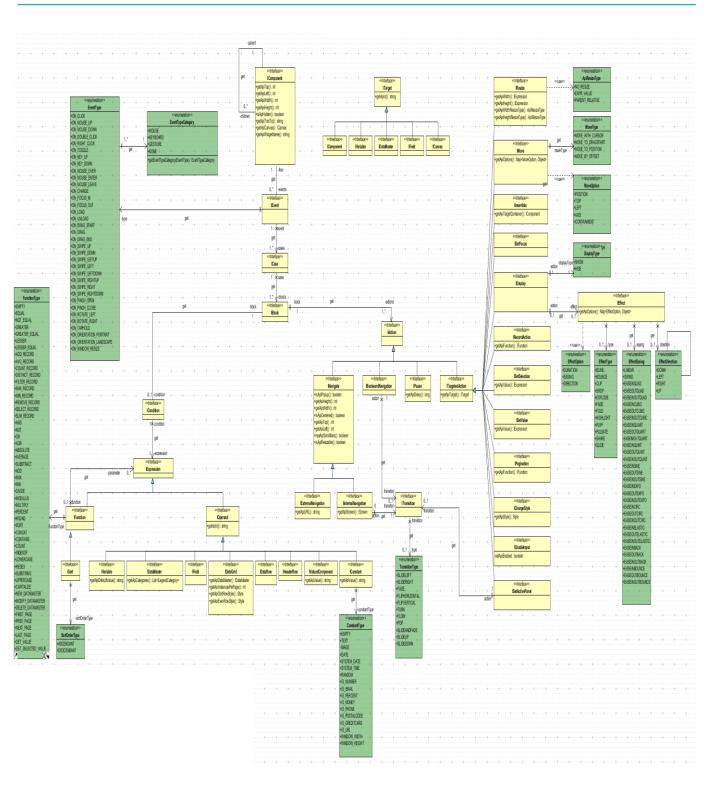


(the diagram continues in the next page)

COMPONENTS CLASSES DIAGRAM (2)



CLASSES REFERRING TO EVENTS





SECOND EXAMPLE: TRANSLATOR

SECOND EXAMPLE: TRANSLATOR

Now that we've seen in detail all the information you can gather from a prototype let's use it to build an example. Imagine we would like to have a MS Excel file with all the texts in your prototype and send it to a translator. We are going to build an example to do exactly that. We will extract all the texts from labels, buttons, links and so on from a prototype and create a MS Excel file with all that information.

First create a plug-in project like the one you did in the first example. Name it 'Translator' and create a main class inside the src folder. Place this code inside the main method:

```
FileDialog dialog = new FileDialog(new Shell());
String path to vp = dialog.open();
String temp_directory = System.getProperty("java.io.tmpdir");
FileUtils.unZip(path_to_vp, temp_directory+File.separator+"work");
IPrototype prototype =
API.getPrototypeLoader().loadPrototype("prototype",temp directory+File.sep
arator+"work");
  List<ICanvas> canvas = prototype.getApiCanvases();
  String content = "Word in prototype\n\n";
  for (ICanvas current : canvas) {
   content+="Screen: "+current.getApiName()+"\n";
   List<IComponent> components = current.getApiRoot().getApiChildren();
   for (IComponent iComponent : components) {
      content+=addTextsToContent(iComponent);
   }
  }
FileDialog dialog = new FileDialog(Display.getDefault().getActiveShell(),
SWT SAVE):
dialog.setFilterExtensions(new String[] { "*.csv" });
dialog.setOverwrite(true);
String path = dialog.open();
if (path != null) {
 Writer output = null;
 File file = new File(path);
  FileWriter fw = null;
  try {
    fw = new FileWriter(file);
    output = new BufferedWriter(fw);
   output.write(content);
    output.close();
    Desktop.getDesktop().open(file);
  } catch (IOException e) {
} else {
 // show error
l
```



SECOND EXAMPLE: TRANSLATOR

Select that class, then right click on it and choose run as java application. You will be requested to select a .vp file and where do you want to save the .csv file with all the texts in the selected prototype. How is this code doing that?

The first piece of code loads the information of a .vp file in the IPrototype class. Then there's another piece of code that browse the elements of each screen, template or master using the class hierarchy described in the 'Component classes diagram' described in the previous section of this document. When it founds a text it adds it to a String which will be later used to create the resulting .csv file.

As you see is extremely easy to read the information of a .vp file. You can use this option to generate any kind of things you like. You could even generate your own HTML code if you like.



EXTEND JUSTINMIND WITH PLUG-INS

EXTEND JUSTINMIND WITH PLUG-INS

So far we have talked about how to create code that is able to read the contents of a .vp file and do something with them. Now we will see how to add menu and tool bar options inside Prototyper itself.

Justinmind Prototyper has been built using a Eclipse Framework technology called RCP (Rich Client Platform). This technology allows you to extend the application's functionality using plug-ins. There's a lot of information available in the web about how to develop plug-ins for RCP applications like Prototyper. What this document will cover are just the basics to build simple plug-ins for Justinmind Prototyper. We encourage you to go deeper on the plug-ins development if you want to build more complex things.

So, let's start! First we need to create a new plug-in project just like we did in the previous examples. Go to file \rightarrow Project \rightarrow Other and select Plug-in Project. Give it a name and push 'next'. Then check the options 'Generate an activator...' and 'This plug-in ...' like in the previous examples but this time instead of the 'Finish' button press 'next'. Then select the 'Hello, World Command' template:

New Plug-in Project Templates Select one of the available templates to gener Create a plug-in using one of the templates Available Templates: Custom plug-in wizard Hello, World Hello, World	This wizard creates standard plug-in directory structure and adds the following: • Command contribution. This template
Plug-in with a multi-page editor Plug-in with a popup menu Plug-in with a property page Plug-in with a property page Plug-in with a reflor Plug-in with an incremental project build Plug-in with an incremental project build Plug-in with an incremental Project build Plug-in with a mincremental Plug-in with a m	creates a simple command contribution that adds Sample Menu to the menu bar and a button to the tool bar. Both the menu item in the new menu and the button invoke the same Sample Action. Its role is to open a simple message dialog with a message of your choice. Extensions Used org.actipse.ui.commands org.actipse.ui.bandlers org.actipse.ui.bindings org.actipse.ui.menus
(?) Sack Ne	st > Finish Cancel

And press 'Finish'. The new project will appear in the project's folder at the left. Open the src folder and look for the class called SampleHandler inside handlers package. That class is the main entry point to your code. Open it and look for the method 'execute'. That it's the equivalent of the 'main' method we had in the previous examples. So write some code there just to test the integration (you can copy and paste the code of one of the examples).



EXTEND JUSTINMIND WITH PLUG-INS

Now you need to configure your plug-in to be able to work in Prototyper. Open the file called 'MANIFEST.MF' that is located inside the META-INF folder. Then select the tab called 'MANIFEST.MF' and add this line at the bottom:

Eclipse-RegisterBuddy: com.justinmind.prototyper.api

And press 'Save'. Now your plug-in and the API are 'buddies' so they will be able to work together and so on. Don't think too much about it, if you want to know what that sentence really does check the information in www.eclipse.org.

Select the tab called 'Dependencies'. Push the 'add' button and write 'com.justinmind'. Select the three plug-ins listed and press 'ok'. Then select 'File \rightarrow Save'. Now you have a project configured to access to all the methods in the API.

CONFIGURE COMMAND NAME

Open the file named plugin.xml and select the tab 'extensions'. Then select the 'Sample Command' node in the extensions tree (inside org.eclipse.ui.commands branch). There you will be able to define the menu option name that will execute your plug-in inside Justinmind.

ADD TO APPLICATION MENU AND TOOLBAR

In order to make your plug-in visible inside Justinmind, you need to set its location within the workbench menu and the toolbar of the application: Open plugin.xml and select the tab 'extensions'. Select 'org.eclipse.ui.menus' node and then 'menu:org.eclipse.ui.main.menu'. Replace its locationURI with the following text: **'menu:org.eclipse.ui.workbench.menu?after=additions'.**

We highly recommend you to append your plug-in to the 'plugins' menu already available in Justinmind so the users of your plug-in know where to find it. So if you want your plug-in to be included inside the 'Plug-ins' menu option in Justinmind once the user imports it, then you need to customize the 'org.eclipse.ui.menus' Extension this way: Select 'Sample Menu' node in the extensions tree and set as $id \rightarrow plugins$ and as $label \rightarrow Plug-ins$ just like the screenshot below:

😪 Extensions				0 - 🎄 - ≉ 🔊
Al Extensions Define extensions for this plug-in in the following section. Type filter test	Up	Set Inb id: id: icco icco icco icco icco icco	the properti e <u>monic:</u> n: (tip: nmandid:	hert Details lies of "monu". Required fields are denoted by "*". Plug-ins plugins Browse Browse Browse
Jverview Dependencies Runtime Extensions Extension Points	build IV	MAINIPEST.MP	piugin.xmi	build.properties



EXTEND JUSTINMIND WITH PLUG-INS

The plug-in is ready now but you need to export it out of the SDK. Select the tab 'overview' and look for the link that says 'Export Wizard':

Image: state of the state					(Dave 12) Differentiation in a	The second
Configure JIPE executions To package and export the plug-in: Update the classoft settings: 0. Organice the plug-in: using the (<u>Deprive Visions' Visio</u>	Package Explorer 12 Stephorer 12 Staff RE System Library (LewSE-1.6) Replays he perpendencies or rc Staff META-3RE Diplayscamp phoginame PrintScreen Names	PrintScreenNames PrintScreenNames PrintScreenNames PrintScreenNames PrintScreenNames PrintScreenNames Schutter this plug-in when one of This plug-in is a singleton Schutter this plug-in when one of This plug-in is a singleton Schutter this plug-in when one of This plug-in is a singleton Schutter this plug-in when one of This plug-in it is a singleton Schutter this plug-in when one of This plug-in it is a singleton Schutter this plug-in when one of This plug-in it is a singleton Schutter this plug-in when one of This plug-in it is a singleton Schutter this plug-in when one of This plug-in it is a singleton Schutter this plug-in when one of This plug-in when one of This plug-in it is a singleton Schutter this plug-in when one of This plug-in it is a singleton Schutter this plug-in when one of This plug-in it is a singleton Schutter this plug-in when one of This plug-in it is a singleton Schutter this plug-in when one of This plug-in when one of	its cleaser is loaded	Translator Translator Browsen Add Remove Up Down	Extension: declars combudion Extension: declars combudion Exting Texting Texting Texting Linch: A RAP Application Linch: A RAP Application Linch: A RAP Application Linch: A RAP Application Texting Textin	a this play-ain markets to the platform. netion point this play-ain adds to net Eclipse application: usy mode Datug-mode gambe Manifest: Ward play-in using the <u>Externalise Strings.</u> el in the deployable play-in on the,

Then select a destination folder to save your plug-in and press finish. Your plugin will be saved in a folder called 'plugins' inside the one you defined as the destination folder.

You are all set and ready to test your plug-in in Justinmind. Open Justinmind and select the option 'Install a plug-in' that you'll find in the Plug-ins menu. Select the plug-in you've just exported and restart Justinmind. Now you will have a new option available inside the Plug-ins menu that refers to the plugin you just made.

So that's all folks! Now it's time to build your own plug-ins and share them (or sell them) with all the other Justinminders around there!





REFERENCES

Eclipse - The Eclipse Foundation open source community website. <u>http://www.eclipse.org/</u>

The Java™ Tutorials <u>http://docs.oracle.com/javase/tutorial/</u>

Stack Overflow : questions about Java development http://stackoverflow.com/



APPENDIX: DEBUG YOUR PLUG-IN'S CODE

APPENDIX: DEBUG YOUR PLUG-IN'S CODE

There is way, if you are using the SDK for MS Windows, to debug your plugins without exporting them. Inside the SDK go to Window \rightarrow Preferences and open the option 'Plug-in Development'. Select 'Target platform' and push the 'add' button. Then push 'next' and the 'add' button again. Select 'directory' and then 'next'. Push 'browse' and select the folder where you installed Justinmind (typically c:\Program Files\Justinmind\Justinmind Prototyper x.x.x) and then press 'finish'. Press finish again and select 'new target' as the new Target Platform:

Preferences	And a state of the second seco	_ D X
type filter text	Target Platform	⇔ ▼ ⇒ ▼
type filter text General Ant Help Install/Update Java Plug-in Development API Baselines API Errors/Warnings Compiles Editors OSGI Frameworks Target Platform Run/Debug Tasks Team Usage Data Collector Validation XML	Add, edit and remove target definitions. The active target d used as the target platform which workspace plug-ins will be tested against. New definitions are stored locally, but they or project in the workspace and shared with others. Iarget definitions:	Reload Add, £dit Øemove Share
	C:\Program Files (x86)\Justinmind\Justinmind Prototyp	er 4.5.0 - C:\Progra
	Restore <u>D</u> efaults	Apply
?	ОК	Cancel

Finally press 'ok'. Now you have your SDK configured and ready. Now if you want to run or debug your plug-in you just have to right click on the project and select 'Run as Eclipse Application'. Then your Justinmind Prototyper will be executed with your plug-in already installed.



225 Bush St. Fl 12 San Francisco, CA 94104-4254 United States www.justinmind.com jim.sales@justinmind.com