SWTBot improvements

Supporting native dialogs

outline

1 - Problem.....3
2 - Proposed Solution.....4
2.1 - Modifications in SWT.....4
2.2 - Modifications in SWTBot.....4
3 - Tasks.....6

1 - Problem

Some of the dialogs that can be opened by Eclipse are implemented through direct OS calls to open the native dialogs provided by the machine’s OS. Some of the basic functions implemented that way include the dialogs to prompt a user to select a file or a folder, choose a color from a gradient, print content, …

These dialogs are under direct control of the OS and out of reach of SWT, preventing SWTBot from interacting with them as would be needed to test their behavior. Furthermore, SWTBot tests will « block » on these dialogs since there is no way to programmatically close them once they’re opened, automatically failing the tests.

The purpose of this study is to propose a way for SWTBot to interact with native dialogs so that they can be tested.

2 - Proposed Solution

Since we lose the capability of controlling the application under test (AUT) as soon as the OS calls are made, we will need to prevent these calls altogether when in the context of SWTBot.

We need to change SWT so that it warns interested parties that it's going to open a native dialog. It will allow these interested parties to prevent the action ("Event.doit" provides us with an existing mean for this). If this boolean is unset, SWT will not give control over to the OS for native dialogs, but instead will open a modal Shell and run its event loop (to prevent users from interacting with the UI until the dialog is "closed" as would normally be the case). This event loop will run until SWT receives another event telling it that the dialog can now be closed, with the expected result sent along with the event (again, Event.data provides us with an existing mean for this), at which point SWT can resume execution as if the dialog had been opened.

In turn, SWTBot will register a global listener to intercept the "dialog opening" events sent by SWT and take over, setting the "doit" boolean to false and providing an API for the test cases to interact with the dialogs. SWTBot will thus not test the dialog themselves -we consider native dialogs not to require tests- but will only concentrate on the SWT parts of the AUT.

This will require modifications in both SWT and SWTBot :

1. SWT will send a notification before it makes the OS calls, allow interested third parties to intercept this event and tell SWT not to give control over to the native dialogs so that they can retain it,
2. SWTBot will listen and react to these notifications, tell SWT not to execute the native calls, set up a blocking Shell to « mimic » the model nature of the dialogs, and finally provide the API for tests to set the necessary result from the dialogs.

The AUT expects a dialog to be opened and only returns a result when the OS does so, all native dialogs being modal. Even if we change the implementation to allow event listeners to prevent the OS calls, we still need to make sure the execution blocks somehow as if a modal dialog were opened, and that third parties have a way of waking us up and provide us with a result to return.

2.1 - Modifications in SWT

All changes will concentrate on the *open()* method shared by the *Dialog* implementations. Two new event types will also be introduced in the *org.eclipse.swt.SWT* class : *NativeDialogOpen* and *NativeDialogClose*.

Right before the OS calls are made, SWT will send an event of type *NativeDialogOpen*. If any of the registered listeners has set the *org.eclipse.swt.widgets.Event.doit* boolean of that event to *false*, the dialog will not be opened. Instead, SWT will run the event loop on the parent shell of the dialog while waiting for someone to notify it with an event of type *NativeDialogClose*. It will expect this new event to hold the result as event data.

While running the event loop, the dialog will open an empty, modal shell in order to prevent user interaction until actually closed.

The issue concerns all of the sub-classes of *org.eclipse.swt.widgets.Dialog* and every individual sub-class will need to be modified separately.

JUnit Tests will need to ensure that we can safely block the opening of the native dialogs and "wake up" the SWT dialog to continue execution.

2.2 - Modifications in SWTBot

SWTBot will register a global listener for all events of type *NativeDialogOpen* to prevent all native dialogs which types are handled from opening and keep references to their instances.

A new kind of bot will be needed for each kind of dialog we're going to support (e.g. SWTBotFileDialog), reflecting the public API of the underlying dialog classes but also enhancing them with the required API to allow tests to set the desired result and close (or cancel) the dialog.

Closing the dialogs will make use of events of kind *NativeDialogClose*, with the expected result as payload.

3 - Tasks

A draft implementation that demonstrates the changes needed in both SWT and SWTBot will be attached to the bugzilla #283609. This draft concentrates on the *FileDialog* and allows users to write tests such as :

```
// Assume we have a project named "test.project" in the workspace
@Test
public void testNativeDialog() {
	SWTWorkbenchBot bot = new SWTWorkbenchBot();

	// Find the project location on disk
	String projectLocation = 
ResourcesPlugin.getWorkspace().getRoot().getProject("test.project").getLocation().toStr
ing();

	packageExplorerView.bot().tree().getTreeItem("test.project").select();
	bot.menu("File").menu("Properties").click();
	bot.tree().getTreeItem("Java Build Path").select();
	bot.tabItem("&Libraries").activate();
	bot.button("Add External JARs...").click();

	// We would normally have lost control of the AUT at this point
	bot.fileDialog().setFilterPath(projectLocation).setFileNames(new String[]
{"plugin.jar", "lib.jar", }).close();
}
```

This allows the user to bypass the native dialog that should have been opened by *bot.button("Add External JARs").click()* and actually set the expected result of the *FileDialog* that this action would have opened without the patch.

What remains to be done if this approach is validated is to extend the modifications this draft makes in *FileDialog* to all other sub-classes of *org.eclipse.swt.widgets.Dialog* :

- AttachDialog
- ColorDialog
- DirectoryDialog
- FontDialog
- GradientDialog
- MessageBox
- PrintDialog

The draft also doesn't implement a blocking Shell to mimic the modal nature of the native dialogs, which mean users could interact with the UI while a Dialog is running its event loop. This will need to be implemented.

Finally, tests have not been written in that draft. The final implementation will at least need to ensure, that :

- A Dialog "blocked" from opening through a *NativeDialogOpen* event's data properly blocks the execution of further code from the dialog's callers
- A Dialog "blocked" from opening through a *NativeDialogOpen* event's data can properly be woken up by a *NativeDialogClose* event, both for the "cancel" and "Ok" paths (with or without "selected file" result)