Frequently Asked Questions

This PowerTools FAQ answers many frequently asked questions regarding the functionality of the various parts of the PowerTools suite. The questions are organized in the following areas:

- General,
- Writing test cases,
- Using the PowerTools with FitNesse, and
- Programming instructions

Please send any questions or suggestions that can help improve this FAQ (or any other PowerTools document) to info@thepowertools.org.

General

Q: What license(s) cover the PowerTools?

A: The PowerTools themselves are released under AGPLv3 (the Affero General Public License version 3). The details of this open source license can be found at http://www.gnu.org/licenses/agpl-3.0.html. The PowerTools engine makes use of the ANTLR parser generator and its runtime binaries are included in its distribution. ANTLR is released under the BSD license that can be found at http://antlr.org/license.html. The full text of both licenses should be included in any PowerTools distribution.

Q: Where can I find the latest version of the PowerTools documentation?

A: At its website www.thepowertools.org, under documentation.

Q: What is the difference between an instruction and a keyword?

A: None, really. The term 'keyword' usually refers to a form of test where the first column contains the action and the remaining ones contain only arguments, which is common when using a spreadsheet for tests. The term 'instruction' is somewhat more general, simply indicating that the statements in an automated test are instructions to the automated testing solution. It is also used more specifically for an action that is written as a sentence, with arguments mixed into it. But this distinction in form really is most relevant inside the engine code that supports both of these forms. Since most people only use one form, the chances of confusion are slim and they can use whatever term they prefer.

Q: How does the engine work?

A: It basically does the following in a loop:

1. Read an instruction from the current input document,
2. Evaluate any expressions in the instruction (i.e. anything starting with a '?'),
3. Find and invoke the implementation of the instruction, passing it the arguments, and
4. Report the results.

More details can be found in the PowerTools engine design document.

Writing test cases

Q: How do I make programmed instructions available to my test?

A: Use the ‘use instruction set’ instruction to register the instruction set class with the engine. All instructions in the instruction set will then be available to the test. The methods that implement instructions must have the proper signature to be found (see below).

Q: How do I select one of several instructions with the same name?

A: It is allowed (although not advised) to have multiple instruction sets that implement an instruction of the exact same name. When such an instruction appears in a test, the engine will detect the conflict and report an error. Select the intended instruction by prefixing the instruction name with the instruction set name, e.g. ’ordering.add client’.

The instruction set name to use is:

- The name specified when registering the instruction set (‘use instruction set’), if a valid name was specified,
- Otherwise, the name returned by the `getName()` method of the instruction set object, if the instruction set class implements the `InstructionSet` interface and `getName()` returns a valid instruction set name,
- Otherwise, the fully qualified class name of the instructions class (e.g. `com.company.product.package.ClassName`).

Using the PowerTools with FitNesse

Q: How do I work with FitNesse?

A: FitNesse is a separate open source test tool with its own documentation. Both the software and the documentation are available through its home website http://fitnesse.org. The answers to any questions that are not PowerTools specific should be available there or through other internet resources.

Q: How do I set up FitNesse to use the PowerTools?

A: To make the PowerTools fixtures available to FitNesse while running a test, the PowerTools jar file and the FitNesse jar file must be placed on the FitNesse classpath. This is normally done by adding the below lines to the root page of your test.

```
!path <pathToPowerToolsJars>/powerTools4FitNesse.jar

!path <pathToPowerToolsJars>/fitnesse.jar
```

Programming instructions

Q: What programming languages can I use to implement instructions?

A: Only Java at the moment. Support for .NET is being added.

Q: How do I create an instructions class in Java?

A: Any class can be registered with the engine as an instructions class, but the general form is:

```
public class SomeInstructionsClassName {
	private RunTime runtime;

	public SomeInstructionsClassName (RunTime runTime) {
		this.runTime = runTime;
	}

	...
}
```

Q: When does an instructions class constructor need the `RunTime` parameter?

A: If any method in an instructions class needs access to any engine functionality, such as reporting methods or the symbol table, then the `RunTime` parameter is required. If the instruction methods only interact with the engine using an `ExecutionException` to report an error, then the runtime parameter (and the corresponding class member) can be omitted. The engine will then use the constructor with no parameters to instantiate the class.

Q: What should an instruction method look like?

A: The method name for an instruction is obtained by capitalizing the first character of each word and then concatenating the words. So the method name for ‘add XYZ order’ is AddXYZOrder (the abbreviation remains as it was). The return type can be `void` or `boolean`. If the return type is `boolean`, a return value of `false` will be interpreted by the engine as indicating a failing instruction and will be reported as an (additional) error. Errors can also be reported through the engine runtime or by throwing an `ExceutionException`, so the return value is not really needed.

If the instruction has one or more parameters, how to deal with these depends on whether or not it concerns a keyword. For a keyword, with all parameters following it on a line, simply list the types and names of the parameters in the desired order. Note that most basic types are supported, not just strings. The engine will try to parse arguments and pass them as the specified type.

If the instruction is not a keyword and parameters can appear in any place, indicate where each parameter goes using underscores. No numbering is needed, but the number of underscores must match the number of parameters.

```
public void AKeywordWithParameters (String x, int y) {
	…
}
```

```
public boolean AnInstructionWithParameters_And_ (String x, int y) {
      …
}
```

Q: When should an instructions class implement the InstructionSet interface?

A: The `org.powerTools.engine.InstructionSet` interface declares some methods that the engine will call:

1. If no instruction set name is provided in a ‘use instruction set’ instruction, the engine uses the `getName()` method to determine its default name. This instruction set name is needed only to distinguish between multiple instructions with the exact same name by prefixing the instruction name with the instruction set name: e.g. ’ordering.add client’. The `getName()` method simply allows the instructions class to offer the tester a more compact name for itself than the default, which is the fully qualified class name.
2. If an instruction set constructor does not fully prepare the instruction set object for use, any remaining steps can be executed in a `setup()` method. The engine will call this method after instantiating the instruction set object.
3. If an instruction set object needs to clean up at the end of a test run, for example by closing a file or a connection, then this can be done in the `cleanup()` method. The engine will call this method before it terminates.

To use any of these mechanisms, it is necessary for an instruction set to implement the `InstructionSet` interface.