Chapter 8

JSystem Automation Framework Advanced Functionality

➢ **In this chapter…**

8.1 JSystem Automation Framework Advanced Functionality Overview

The JSystem Automation Framework advanced functionality chapter details the JSystem Automation Framework and drills down into the application explaining the flexible and dynamic elements that make up the unique and highly advanced automation testing framework, detailing the architecture of the JRunner and understanding how to increase control over scenario execution as well as issues relevant to large scale projects.

8.1.1 Controlling Executions

When performing an execution of a scenario, the JRunner JVM spawns a child JVM that executes the scenario. The child JVM activates the "**ant**" interpreter and passes it the interpreter to the scenario file path.

After spawning the JVM the JRunner opens a server socket and waits for the tests JVM to open and make a connection to the JRunner. Once scenario execution starts, the tests JVM opens a connection to the JRunner and synchronization messages and reporter messages pass between JVMs.

The classpath of the child JVM is set to be the classpath of the JRunner JVM, the ant that the tests JVM works with can be found under the directory "**JRunner install/thirdparty/ant** ".

8.1.2 JRunner Run Modes

In order to conserve computer resources while executing scenarios the JRunner JVM and the test execution JVM have been separated.

The JRunner has three execution modes:

- **Run Mode 1** - The default run mode where one JVM is created for the whole scenario execution.
- **Run Mode 2** - A new JVM is created for each test.
- **Run Mode 4** - A new JVM is created for each sub scenario.

"**Run Mode 2**" was developed to perform very long test and scenario runs that can exhaust the test JVM and cause an execution failure, thus a new JVM is spawned for each test. As a result, when implementing "**Run Mode 2**" special considerations must be made for extended running time as well as the fact that data cannot be shared using JVM heap space functionality.

In order to relieve these two limitations "**Run Mode 4**" was developed, in this run mode a JVM is allocated for each sub-scenario, so that tests in the same sub scenario can share the JVM heap space and execution, greatly increasing test execution speed.

In order to configure a "**Run Mode**" set the property "**run.mode**" in "**JSystem properties**".

In the default "**Run Mode 1**" do not write anything or write "**Run.mode =1**", when using "**Run mode 2**" write "**run.mode=2**", and when using "**Run.mode 4**" write "**run.mode=4**".

8.1.2.1 Test JVM Parameters

In many cases the user requires control of the java parameters that are passed to the test JVM. The most common use for this is to change the default memory allocation or to enable remote debugging of the test JVM.
In order to set the test JVM parameters add and or alter the **"test.vm.params**" properties in the "**jsystem.properties**" file.

The two following example are provided:

- In order to alter memory allocation**: test.vm.params=-Xms32M –Xmx1024M.**
- To perform a debugging test execution: **test.vm.params= -classic -Xdebug -Xnoagent -Djava.compiler=NONE -Xrunjdwp:transport=dt_socket,address=8787,server=y,suspend=y|**

8.1.2.2 JRunner Classpath

Automation projects with JSystem are Java projects, like all java projects when the project grows, additional libraries (jars) need to be added, and the order of the jars becomes important to avoid and understand conflicts.

8.1.2.2.1 Jars Order in a Class Path

The class loader of the JRunner loads jars from the following libraries in the following order:

1. JRunner/thirdparty/ant/lib
2. JRunner/thirdparty/commonLib
3. JRunner/thirdparty/lib
4. JRunner/customer_lib
5. JRunner/lib
6. user additional libs (see next section)
7. automation project lib (lib folder in test project)
8. automation project tests

8.1.2.2.2 Adding User Jars

In order to add "**jars"** that do not reside in one of the above folders, use the property "**lib.dirs**" in the "**jsystem.properties**" file. Jars should be separated with a semicolon.

For Example: lib.dirs=c:/additionaljars/myJar.jar; c:/additionaljars/mySecondJar.jar

8.1.2.3 Shutdown Hooks

In order to add threads that will be invoked when the test JVM terminates and when the JRunner exit using the property "**shutdown.threads**" in the "**jsystem.properties**" file.

For Example: shutdown.threads=com.ignis.ShutDownThread;com.myproject.ShutDown2

8.1.2.4 Output Redirect

By default, all JRunner output is dumped in the JRunner console. In order to redirect the output of the JRunner to a file use the property "**stdout.file.name**" in the "**jsystem.properties**" file.

For Example: stdout.file.name=c:/temp/myOutputFile.txt

In order to disable the dumping of JRunner output to the consol set the property to "**console.disable**" in the "**jsystem.properties**" and the file to true.

8.1.2.5 Reducing Memory Consumption in Long Runs

During scenario execution, a summary of the test execution is stored in the JRunner heap for each test that is executed. When a long run is performed this might cause the JRunner to halt.

In order to disable summary information accumulation set the value of the "**summary.disable**" property in "**jsystem.properties**" file to true.

8.1.3 Distributed Execution Plug In

This plug in is soon to be included in the next JSystem Release.

8.1.4 Scripting Language Plug In

This plug in is soon to be included in the next JSystem Release.

8.1.5 Working with Source Control

Guidelines for managing the JSystem automation project code with source control:

1. JSystem scenarios and SUT files should be treated as code, in terms of their source control management. At the beginning of a project this guideline seems annoying but when the number of people that work on the project grows, managing these file in source control becomes critical.
2. Share the **".jsystembase"** file and not the "**jsystem.properties**" file. The "**jsystem.properties**" file includes both internal system information (the path to the classes' folder) and general automation configuration information that should be shared between users (for example: the run mode).

Note: Sharing the "jsystem.properties" file between users can cause the JRunner to behave in an unstable manner.

All the project configurations should be set in a file called .jsystembase. When the JRunner launches JSystem, if the **".jsystembase"** file exists and "**jsystem.properties**" does not exist, the "**jsystem.properties**" file is created with all the properties that are defined in **".jsystembase".**