Rational Functional Tester - Tips and Tricks

Suma Byrappa

Swathi Rao

IBM Rational
Agenda

- Functional Tester - Overview
- Functional Tester – Tips and Tricks
- Demos
- Q&A
IBM Rational Functional Tester

- Formerly known as “Rational XDE Tester”
- Automated functional and regression testing tool for Web based, Java, .NET, Siebel, Terminal based and SAP (coming soon) applications

**Key Benefits**

- Minimize test maintenance with scripts that are resilient to application changes
- Wizard enhanced automation to speed test creation for the *new user*
- Powerful scripting language and IDE for the *professional tester*
Functional Testing – 3 Step Automation Process

Record, Enhance, Execute

- Recorder
  - Test scripts are recorded on the fly, as user navigates application
  - Verification Points and Data Driven Tests can be added while recording

- Scripts
  - RFT uses VB.Net or Java to create easy to understand object-oriented scripts
  - Scripts can be enhanced to add conditional branching, refactoring, datapooling etc.

- Playback
  - Scripts are executed
  - Discrepancies are logged
Verification Points

- Automated Validation
  - Functional Tester captures data and properties that can be invisible to users
  - During script execution, current results are compared to stored baselines
  - Discrepancies are flagged and reported to user in test log

Functional Tester Sees Object Data

Functional Tester Sees Object Properties

You See controls or objects
Dynamic Data/Content Matching

- Use pattern matching technique to verify dynamic data and create robust tests
- E.g. Instead of validating “Order ID 230”, validate “Order ID ####” or Order ID 2##
- This allows for a wide variety of acceptable responses as well as restrictions on acceptable responses when validating the application’s behavior
Data Driven Testing

- Reuse the same script to replicate different (realistic) scenarios
- Abstracts the test data from the test script
  - Modify test data without affecting the test script
  - Add new test cases by modifying the data and not the test script
  - Share the test data with many test scripts
- Wizard driven process
  - No programming involved
  - Import data from external sources
Object Maps

- Static Hierarchical Representation of the Application Under Test (AUT)
  - Static – includes all relevant TestObjects in AUT, not timing-sensitive.
  - Hierarchical – strict hierarchy, no cyclic dependencies, no indirect associations
  - Representation – maintains recognition properties that “describe” each TestObject

- Eases script maintenance
  - If object in the AUT changes, edit the object in the map
  - All references to object in script will have the updated info
Script Assure™

Version 1.0

Tester Sees

Version 2.0

Customer Log On

User Name:

Password:

No User Intervention Required With ScriptAssure™

Determines Match
Demo
Agenda

- Functional Tester - Overview
- Functional Tester – Tips and Tricks
- Demos
- Q&A
Tip 1 : Controlling RFT Playback – STOP / PAUSE

- Use F11 key to perform a **controlled** stop
  - Closes out the log and cleans up before stopping
  - Use `stop()` command to control where stop occurs
- Use F12 key to **pause** playback
- Don’t like F11 or F12 well you can change it
  - In `{install}/ivory.properties` file change property
    rational.test.ft.script.playback.stop.hotkey=122
    rational.test.ft.script.playback.pause.hotkey=123
- Stop, Pause, Resume from the Playback Monitor
  (Beware that moving the mouse during playback may effect script execution)
  - Terminate playback using the stop button
  - Pause playback using the pause button
  - Resume playback using the Resume button
Tip 2: Know Those Functional Tester Preferences

- Define settings for how you want the workbench, compiler, Functional Tester, etc. to work.
Tip 2 : Know Those Functional Tester Preferences

- Show Script Launch
- Display Log
- Prompt for Overwriting
- Log Type
  - HTML
  - Text
  - TPTP
  - TestManager
Tip 3: Test Your Enablement

- Especially HTML!!!

Note: .Net and Windows applications do not require enablement.
Tip 4 : Test Object Inspector

- Is the SUT ready to test?
- What properties does it have?
- What is the class and object hierarchy?
- What methods are available to invoke?

- Find out with a wave of the cursor
  - Test Object under the cursor is auto-magically explored

- Use Copy / Paste to capture state of a Test Object in the SUT
Tip 5: **Coming Soon** Clipboard VP Support

- Select *Script Support* ♻ button on the Record Toolbar
Tip 6: Data-Driven Testing

- Scripts already have a Datapool asset
  - Private – associated with one script
  - Public – associated with zero or more scripts

- Population of a Datapool:
  - From a CSV file when created
    - The CSV can be from existing TestManager Datapool
  - Better yet let’s record it …

---

Demo
Tip 6: Data-Driven Testing Notes

- No loop in script
  - Managed by script `initialize, terminate`
    & `callScript` by default
  - Can manually control Datapool iteration using
    `DatapoolFactory.get().load(<file>)` to fetch the desired Datapool
- Recorder dynamically populates the Datapool
- Playback wizard includes `Iteration Count` field
- Verification Points may contain Datapool references
Tip 7: Private vs. Shared Object Maps

**Private**
- Isolated from others
- Good for development
- May lead to redundancy

**Shared**
- Shared between multiple scripts/multiple testers
- Single point of maintenance
- Can have Private maps merged into it
Tip 7: Object map editing flexibility

- Script Maintenance can outpace script development as the volume of tests grows
- Functional Tester includes an Object Map update tool which enables batch updates to a centralized object map
  - Modify Properties or Values of objects
  - Reduces time spent fixing individual scripts
  - Frees up more time for script development
Tip 7: Troubleshooting Object Recognition Failures

- If object can’t be found:
  - Is the SUT enabled & testable?
  - Recognition properties are not resilient or
  - The ancestry structure has changed, or
  - Recognition properties have changed in the parent object(s)

- To fix object recognition:
  - Update Recognition properties in the OME
  - Use Regexp for object property values. For Eg:
    - AUIML objects have randomly generated object .id properties that change with every login: W047RX7389, W37AK3896
    - The .id property value can be regexped to (W.+)
  - Adjust recognition property weight to favor more resilient properties
Tip 7: Troubleshooting Object Recognition Failures

- Object Map Editor
  - Start an Application
  - Highlight Test Objects
  - Insert Test Objects and Unify them
  - Update Test Object
  - Convert to Regular Expression (& Evaluate it)
  - Adjust Property Weight
Tip 8: Scripting Tips – IDE Support

- Eclipse and VS.Net are developer *dream* IDEs
  - Lots of capabilities, see preferences, IDE docs
- All time favorite is intellisense support …

- Position Cursor
- Ctrl-Space
- Start typing to shorten list
- Choose Method
- Note the methods signature
Tip 8: Scripting Tips – RFT Support

- **RFT Scripting toolbar**
  - Record Support in the IDE
    - Insert VP, Insert Test Object, Insert Data Driven Commands
  - Find Literals and Replace with Datapool References
- **Context menus in Script Explorer pane**
  - **Insert At Cursor** – on a VP
    - Remember that VPs are context sensitive and will not work anywhere
  - **Insert At Cursor** – on a Test Object
    - Use **Highlight** to make sure the Test Object is active at the right time
  - **Interface Summary** – on a Test Object
    - What methods, data types, recognition properties are available for a particular Test Object. Lots of things not available via the recorder.
Tip 9 : Extending the Screen Snapshot Support

- Screen Snapshot taken only on fatal script errors
- Log yourself using RationalTestScript methods:
  - `logInfo(String, BufferedImage)`
  - `logWarning(String, BufferedImage)`
  - `logError(String, BufferedImage)`
  - `getRootTestObject().getScreenSnapshot()`
  - `[Any GUI Test Object].getScreenSnapshot()`

- For example:
  - `logInfo("Script End", getRootTestObject().getScreenSnapshot());`
  - `logInfo("About to click OK Button", okButton().getScreenSnapshot());`

- Automatically snapshot application state at script end by inserting into Helper Superclass with overload of onTerminate() method
  - Constrain to top level script using isMainScript() method
Tip 10: Event Handling

- Events handlers are called automatically by the RationalTestScript class
  - onInitialize - Called at the start of every script
  - onTerminate - Called at the end of every script
  - onAmbiguousRecognition – Called if more than one matching object is found
  - onObjectNotFound – Called when an Object is not found during playback
  - onSubitemNotFound - Called when a sub item cannot be found.
  - onUnhandledException – Called when unknown exceptions are thrown
  - onVpFailure – Called when the specified VP fails to compare successfully

- Override these methods to handle these events
  - NOTE: Do not call these methods directly
Tip 11: Test Object Anchors & State

- Specified in the Script
  - `process = startApp("ClassicsJavaA");`
  - `OkButton(process, DISABLED).click();`

- Anchor – Test Object from which the search should start
  - Tip 0.5 – Use ProcessTestObject as the anchor

- State
  - Enabled, Showing & Ready is the default
  - Enabled
    - Pre-6.1 – Test Object ignored in find if Enabled state wrong
    - Post-6.1 – If best candidate is Disabled and looking for Enabled then **wait** to see if best candidate becomes Enabled
  - Ready – Browser specific, waits for page to be fully rendered
Tip 12: You Trying to Pick an Argument?

Passing data with callScript

- By default, automated callScript does not include arguments
- RationalTestScript API overloads callScript with arguments

```java
protected java.lang.Object callScript(
    java.lang.String scriptFullName,
    java.lang.Object[] args)
```

```java
callScript("TestCallExternal");
```
Tip 12: Called Script Already Takes Arguments

public class TestCallExernal extends TestCallExernalHelper
{
    public void testMain (Object[] args)
    {
        // ...
    }
}

- Add code to handle args and optionally, return value

public class TestCallExernal extends TestCallExernalHelper
{
    public int testMain (Object[] args)
    {
        for (int i = 0; i < args.length; i++) {
            System.out.println(args[i]),
            // ...
        }
        return -1;
    }
}
Tip 13: Cross-Platform Use of Native Controls

- Using the Windows control support does not work on Linux
  - Nested native controls in Java do not get recorded
- If interested in Linux playback or Native control support in a Java application use
  - Best solution is to use TestObject.find()
  - RationalTestScript.getTopWindows() returns IWindow[]
  - RationalTestScript.getScreen() returns IScreen
- Use IWindow interface methods
  - getChildren() 
  - getText()
- IScreen interface methods include:
  - getActiveWindow()
- See API on-line docs for rest
Tip 13: Example of File Download Dialog

```java
IWindow[] children = getScreen().getActiveWindow().getChildren();

for (int i = 0; i < children.length; ++i) {
    if (children[i].getText().equals("&Open")) {
        children[i].click();
        break;
    }
}
```

Get children of active window

Find object with text "&Open" and click it
Tip 14 : Command-Line Support

- Executing scripts from a command line allows you to integrate Functional Tester with external test drivers, such as:
- Start enabler and application configuration tools from the command-line to initialize test environment without raising the IDE
- Display IDE neutral Verification Point and Object Map editors for a quick fix
- Script creation using the recorder or just creating an empty script
Tip 14: Command-Line Script Execution

- To play back a Java script (order of args is significant):
  ```
  java -Drationa_ft.install.dir=<Rational FT install directory> <-classpath...>
  com.rational.test.ft.rational_ft -datastore <directory> -log <logname> [options] -playback <script-name> [-args <values>]
  ```
  -Drational_ft.install.dir is going away in the next release

- To play back a VB.Net script:
  ```
  rational_ft.exe -datastore <directory> -log <logname> [options] -playback <script-name> [-args <values>]
  ```

- Use Java command line to record, enable, configure applications and perform other actions

- See help file “Functional Tester Command-Line Interface” in your favorite IDE
Tip 15 : Using the Debugger

- Functional Tester uses full-featured IDE based debugger
- With all those features comes a little complexity
- Best illustrated with a…

Demo

- Focus on Eclipse debugger for this demonstration
Agenda

- Functional Tester - Overview
- Functional Tester – Tips and Tricks
- Demos
- Q&A
Questions
Thank You

Suma Byrappa  
(suma.byrappa@in.ibm.com)

Swathi Rao  
(swathi.rao@in.ibm.com)