RRAFS101 : Rational Robot Automation Framework Support

Welcome to the RRAFS Tutorial
Rational Robot Automation Framework Support

Welcome to our introductory course on installing and using the RRAFS framework. This course goes beyond the most basic activities of installing and using the Rational Robot-based framework and includes enabling the SAFS Framework and the SAFS Engine for Rational XDE Tester.

(This tutorial uses assets available in *RRAFSRelease2004.10.31* or later. Rational Products V2001, V2002, and V2003 should all be supported.)

Goals and Expectations

RRAFS101 Definitions

Goals and Expectations

Before we get into the meat of the tutorial we want to give an overall summary of what the lessons in RRAFS101 offer. This also provides a swell indication of what the lessons will not be offering!

RRAFS101 provides an Automator familiar with Rational Robot a comprehensive introduction to the RRAFS Framework, the SAFS Framework tools, and even provides information on using the SAFS Engine for Rational XDE Tester with RRAFS. There is a huge amount of information crammed into these lessons.

What the course cannot offer is an even deeper level of instruction on highly advanced topics for RRAFS, the SAFS Framework, and the SAFS Engine for XDE Tester. Each of these subject areas could span an entire tutorial on their own! Perhaps future tutorials will go into the more advanced topics if interest is sufficient.

Finally, we want to reiterate that we do assume the Automator is already familiar with Rational Robot. We do not go into extensive detail on using Robot in general, or on navigating the Robot development environment (IDE), but a novice Robot user may be able to follow along and figure things out.

RRAFS101 Definitions

A

App Map: The nickname for an Application Map.

Application Map:

An Application Map is our centralized, single-point of maintenance storage mechanism for Window and Component definitions -- also called recognition strings. Application Maps are opened and processed by SAFSMAPS to make them globally available to all testing tools in the environment.

The Application Map is where we "map" simple user-defined component names to the complex recognition information needed by the underlying automation tools. This is also where we can store predefined Application Constants that will be referenced by the SAFSVARS variable service when necessary.

"App Map" is a common nickname referring to an Application Map.

Application-Independent: The framework or tool is not exclusively designed to work with just one application. It is intended to work across many applications, or all applications, without modification. Simply, it was made to test all applications, not just one.

AUT:

Application Under Test

The application we are testing.

Automator: See "Test Automator"

D

DDE_RUNTIME:

The directory in which the RRAFS Framework is installed. This is not a user configurable location. The RRAFS Framework *must* be installed into the sqabas32 subdirectory where Rational Robot itself was installed. By default, this is:

C:\Program Files\Rational\Rational Test\sqabas32

Designer: See "Test Designer"

Driver:

The *Driver* is generally the tool, class, or entity that is responsible for the overall execution of the test. The Driver opens and reads our test tables, keeps track of test status and results, and performs the parsing and routing of test records to the various Engines for execution.

Some tools -- like RRAFS and WRAFS -- are standalone execution "Engines". They provide both the Driver and the Engine for execution. Other tools -- like SAFSDRIVER -- are strictly Drivers and require the use of external Engines to complete a fully functional testing framework.

E

Engine:

In a pure sense, an Engine is used to execute the instructions retrieved by the controlling test Driver. For example: the "RobotJ Engine" refers to the Engine developed for Rational RobotJ (XDE Tester). This is strictly an "Engine" that requires an external Driver to function.

However, the term Engine often refers to a complete testing framework including both Driver and Engine. For example: the "RRAFS Engine" refers to the complete package of Driver and Engine developed for Rational Robot. Similarly, the "WRAFS Engine" refers to the complete package of Driver and Engine developed for Mercury WinRunner. Both of these implementation have inseparable Driver and Engine components.

F

Functional Scripting: The test technique in which automation tool scripts, or script libraries, are broken down into callable functions or subroutines. These functions provide simple reusable actions or features that can be called many times by external scripts or script libraries, usually with different parameter values to provide variation on the function outcome.

Functional Tester:

This is the "will be known as" name for what used to be called Rational RobotJ, and is currently called IBM Rational XDE Tester. This is an Eclipse-based functional test automation tool for Java and Web environments sold by IBM Rational Software.

See: RobotJ for more information.

I

Input Record: See: "Test Record"

K

Keyword-Driven: A test technique in which tests are expressed with user-defined keywords and actions. A test interpretter is coded or trained to evaluate these user-defined expressions and perform the desired activities. These tests are usually written external to the testing tool. For example, they may be in text files, spreadsheets, or database tables.

R

RobotJ:

This is the "formerly known as" name for what is now called IBM Rational XDE Tester, or IBM Rational Functional Tester. This is an Eclipse-based functional test automation tool for Java and Web environments sold by IBM Rational Software.

This is also the nickname of the SAFS Engine developed for this tool. The RobotJ engine is the first "pure" engine developed with the SAFS Framework. It has no Driver and, thus, requires some external Driver to handle the overall test. Rational Robot is usually the Driver used in conjunction with the RobotJ engine.

RRAFS:

Rational Robot Automation Framework Support

Or more correctly: the SAFS Engine for Rational Robot. This was the first, and is the most complete keyword-driven automation framework offered by SAFSDEV. It is considered to be the "reference implementation" for all SAFS Engines. It is a standalone Engine containing both Driver and Engine implementations. However, RRAFS is also capable of being the controlling Driver calling multiple external Engines.

Often, the term RRAFS is intended to infer the inclusion of Rational RobotJ or Rational XDE Tester -- which is a separate SAFS Engine -- but often readily available to Rational Robot users.

S

SAFS:

SAFS -- Software Automation Framework Support

This is a collection of multi-platform tools and protocols designed to facilitate software test automation. The intended focus of these tools is to augment test automation across many different automation tools. It also defines standard processes and protocols for disparate tools to share data and information.

The SAFS tools -- often referred to as the SAFS Framework -- are used throughout the keyword-driven test automation frameworks developed and released by SAFSDEV on SourceForge at http://safsdev.sourceforge.net

See also Driver, Engine, and SAFS Engine.

SAFS Engine: See "Engine"

SAFSINPUT: SAFSINPUT is the STAFservice provided by the SAFS Framework to handle machine-global file input. The files opened in SAFSINPUT are available to all STAF-enabled testing tools in the environment. However, generally, these files are most often the test tables opened by the active Driver in the environment.

S

SAFSLOGS: SAFSLOGS is the STAFservice provided by the SAFS Framework to handle machine-global test logging. Initialized logs in SAFSLOGS are available to all STAF-enabled testing tools in the environment. All testing tools are expected to log to this service so that the sum of all messages from all tools forms a single test log.

SAFSMAPS: SAFSMAPS is the STAFservice provided by the SAFS Framework to handle machine-global Application Maps. The Application Map data and Application Constants stored in SAFSMAPS are available to all STAF-enabled testing tools in the environment. This service can work in conjunction with SAFSVARS to provide those Application Constants as available variables.

SAFSVARS: SAFSVARS is the STAF service provided by the SAFS Framework to handle machine-global variables. The variable data stored in SAFSVARS is available to all STAF-enabled testing tools in the environment. This service can work in conjunction with SAFSMAPS where the user can specify predefined variable values often called "constants" or "application constants".

STAF:

STAF -- Software Testing Automation Framework

STAF is an open source test automation framework managed by IBM designed primarily for distributed testing across multiple machines and platforms. STAF provides enormously powerful automation utility via an ultra-simplistic interface.

This simple interface is accessible to many different programming languages on many different platforms. It allows all of these disparate environments to communicate, coordinate, and synchronize test automation in a distributed environment.

Visit the STAF Homepage on SourceForge.

T

Test Automator:

The test automator develops and maintains successful automation based on the test designs provided by test designers, and the automation tools used to execute the tests. Designing tests generally consumes more time than automating, so test automators can usually support multiple test designers.

Test Design:

A test design generally contains the flow of desired test progression, user or system input, and an expected result where applicable. One might say that a test design is "a test"; and test designs are "the collection of tests".

In SAFS these are also often called "Test Tables".

T

Test Designer: A test designer generally defines or creates test designs--the tests. They focus on the scenarios for testing the application, and expressing those in test designs. Test Designers are not concerned with automation tool details, languages, or complexities. That is the role of the Test Automator.

Test Framework: The collection of associated tools, technologies, and predefined processes deployed collectively as a system to accomplish test automation.

Test Record:

Loosely, a test record is an individual "line" or "row" of a test table. It generally contains a single command or action to execute. This is also known as an "input record".

Strictly, this is a specific *recordtype* in a test table designated in field #1 by the value of " T ", " TW ", or " TF ". The success or failure of this record type increments "test" pass/fail counters instead of "general" pass/fail counters.

Test Table:

A SAFS test table generally contains the flow of desired test progression, user or system input, and an expected result where applicable. These are the test records that define what is to be done.

One might say that a test table is "a test"; and test tables are "the collection of tests".

Generically also called a "Test Design".

Test Technique:

The physical methods by which test automation is expressed or captured. The most common of these would be *Record and Playback*, *Functional Scripting*, and *Keyword-Driven* methods.

Tester: See "Test Designer"

X

XDE Tester:

This is the "also known as" name for what used to be called Rational RobotJ, and may soon be called IBM Rational Functional Tester. This is an Eclipse-based functional test automation tool for Java and Web environments sold by IBM Rational Software.

See: RobotJ for more information.

Lesson

1 Review Test Designer and Test Automator Roles

A key benefit of RRAFS is that it enables the separation of drastically different expert roles. A good test designer capable of devising well-thought test cases need not have the programming prowess required by test automation tools, and they often do not have the time to do both. On the other hand, the test automation expert devoted to effective automated tests may not have the time or application domain knowledge to devise good application-specific testcases.

Let's spend a little time reviewing these different roles and find out how RRAFS allows these different experts to focus on their expertise.

Separation Can Be A Good Thing

Quiz: Test Designer and Automator Roles

Separation Can Be A Good Thing

The Double-edged Sword of Tests and Test Automation

The most important key to accurately determine the quality of an application is to define and execute a good set of efficient tests. Note, this doesn't specify a good set of efficient *automated* tests. Simply, we need a good set of tests to accurately determine the quality of the application.

Designing a good set of efficient tests is no simple task. It requires a thorough analysis of the application, what the target user is hoping to accomplish, and maybe even a little expertise on "the way things should be" for the type of application involved. This can easily be a full-time job on a complex application. This is especially true if we are starting a new project and no such tests exist. As should be expected, designing these tests takes a specific brand of expertise.

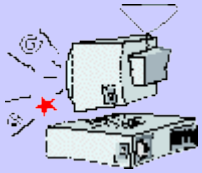
Test automation is a complex software development activity. It requires the mastery of complex development tools and programming languages. The application being tested is a mere subset of the complete complex system needed to automate the tests. Like any software development activity, problems have to be solved. Test code must be debugged. The test code that will automate the testing of the application must itself be tested and enhanced for accuracy and reliability. Like good test design, the software development and maintenance that defines test automation is a full-time job and it takes a specific brand of expertise.

As should be readily apparent, good test design and good test automation are two completely different things that require two completely different types of expertise. Deciding which tests can be or should be automated is a separate issue that will be decided after good tests have been defined.

Jack of All Trades? Or an Expert of None?

The problem? The bulk of today's test automation tools do not effectively provide for the separation of these Designer and Automator roles. They generally expect the tests to be defined and expressed using the programming model or language within the complexity of the test automation tool. They accomodate Automators very well, but do not accomodate Designers much at all.

Consequently, good test Designers are often tasked with trying to automate their tests. More often than not, good test Designers will fail miserably in both designing their tests and attempting to automate them. They usually have neither the time nor the expertise to do both fully and effectively.



The converse is just as true. A good test Automator can do no better and more often than not will fail just as miserably for the same reasons. They usually have neither the time nor the expertise to do both fully and effectively.

It is this model of one-size-fits-all for test automation that most often fails. It is only when we effectively separate these two roles and activities that we can give each the adequate attention they deserve.

Let's review how RRAFS was designed from the very beginning to accomodate this separation of roles.

Test Design in RRAFS

As mentioned, designing a good set of efficient tests is no simple task. It requires a thorough analysis of the application, what the target user is hoping to accomplish, and maybe even a little expertise on "the way things should be" for the type of application involved. This can easily be a full-time job on a complex application. This is especially true if we are starting a new project and no such tests exist. As should be expected, designing these tests takes a specific brand of expertise.

RRAFS accomodates these concepts for Designers rather simply. First, within some comfortable guidelines, a test Designer is allowed to express the test using an action vocabulary of their own design. This is most effective when the Designer expresses each test as an abstract set of application use-cases using a language or vocabulary appropriate for the application.

T	LaunchMyApp		
T	LoginAs	^user="Johnny"	^pwd="BGood"
T	AddClient	^name="Attila D. Hun"	^service="Ballroom Dancing"
T	VerifyClientExists	^name="Attila D. Hun"	
T	DeleteClient	^name="Attila D. Hun"	
T	VerifyClientGone	^name="Attila D. Hun"	
T	LogOff		

The Designer does not have to go into detail of how these things are accomplished. They are tasked to design a good test without regard to whether the test will be done manually or automatically. They use their own vocabulary, they define their own variables or data as necessary. More importantly, there is no hint of any specific automation tool.

The Designer develops these tests in an environment more conducive to designing tests. This may be a simple text editor, a spreadsheet program, or some other more elegant tool provided specifically for RRAFS test development like the SAFS TestGenerator. They do NOT develop the tests inside the programming environment of some test automation tool IDE.

In fact, a Designer can develop tests without ever knowing which automation tool, or which collection of automation tools will actually be used to run the tests! It might be IBM Rational Robot now, IBM Rational XDE Tester tomorrow, something completely new 6 months from now, or all of them at the same time! It doesn't matter, that is the problem the Automator will deal with...

Test Automator in RRAFS

Test automation is a complex software development activity. It requires the mastery of complex development tools and programming languages. The application being tested is a mere subset of the complete complex system needed to automate the tests. Like any software development activity, problems have to be solved. Test code must be debugged. The test code that will automate the testing of the application must itself be tested and enhanced for accuracy and reliability. Like good test design, the software development and maintenance that defines test automation is a full-time job and it takes a specific brand of expertise.

Fortunately, RRAFS has already designed, coded, and (mostly) tested a large collection of tools, reusable code, and implemented keywords that we Automators will never have to duplicate! There are still complicated automation problems to solve and test execution issues to be debugged, but the majority of the true code that needs to be developed is already there.

The primary job of the Test Automator in RRAFS is to take each abstract test design developed by the Designer and provide a concrete implementation. Essentially, the Automator provides the "how" for each abstract action the Designer has defined. And they do this using the tools and the collection of predefined keywords or actions already implemented within RRAFS.

An example concrete implementation of the Designer's abstract "AddClient" use-case we saw earlier:

AddClient ^name="Attila D. Hun" ^service="Ballroom Dancing"

Concrete AddClient Implementation in RRAFS:

T	ClientAdminWin	ClientField	SetTextValue	^name
T	ClientAdminWin	ServiceList	SelectTextItem	^service
T	ClientAdminWin	AddButton	Click	

The Automator focuses their expertise on mapping the application components to be tested to the automation tool(s) that will execute the tests. They ensure the automation does what the Designer intended. They maintain the accuracy and reliability of the tests. They may even add new features, functions, or action commands to the framework to accomodate the needs of the tests.

Predefined Keywords

http://safsdev.sourceforge.net/sqabasic2000/RRAFSReference.htm

Quiz: Test Designer and Automator Roles

The Test Designer and Test Automator roles require drastically different expertise. Let's see how much you have retained from this lesson!

1 1 Marks
The most important key to accurately determine the quality of an application is to define and execute a good set of *automated* tests.

Answer: ○ True ○ False

2 1 Marks
Select 3 items that are *essential* tasks of the Test Designer.

Answer:
- ☐ a. Programming in the application's development language
- ☐ b. An understanding of the way the application "should" work
- ☐ c. Analysis of what the target user hopes to accomplish
- ☐ d. Thorough analysis of the application features

3 1 Marks
Select 3 items that are *essential* skills for the Test Automator role.

Answer:
- ☐ a. Understanding the application "system" including automation tools
- ☐ b. Mastery of development tools and programming languages
- ☐ c. Complex software development activities
- ☐ d. Knowledge of what the target user hopes to accomplish

4 1 Marks
When a Test Designer with little or no Test Automator skills is tasked to do both test design and automation what outcome is most typical?

Answer:
- ○ a. Thoughtfully designed tests are automated poorly.
- ○ b. Poorly designed tests are automated poorly.
- ○ c. There is neither the time nor the expertise to do both effectively.

5 1 Marks
A Test Automator with little or no expertise with the domain or subject matter of the application asked to do both test design and test automation will most often provide these typical results.

Answer:
- ○ a. There is neither the time nor the expertise to do both fully and effectively.
- ○ b. Poorly designed test will be automated well.
- ○ c. Thoughtfully designed tests will be automated well.

6 1 Marks
RRAFS provides for the separation of roles by providing these 2 Test Designer vocabulary benefits.

Answer:
- [ ] a. Designer defines an application specific testing vocabulary (keywords)
- [ ] b. Designer defines their own testing vocabulary (keywords)
- [ ] c. Designer implements vocabulary keywords within the automation tool

7 1 Marks
The Test Designer does not have to provide details of "how" a particular use-case action or keyword is supposed to be accomplished.

Answer: ○True ○False

8 1 Marks
The Designer does NOT develop the tests inside the programming environment of the test automation tool IDE.

Answer: ○True ○False

9 1 Marks
Select 2 key benefits RRAFS provides the Test Automator in regards to the testing vocabulary or keywords used in test designs.

Answer:
- [ ] a. The "how" of Designer keywords is implemented thru Automator keywords
- [ ] b. The Automator must implement Designer keywords in scripts
- [ ] c. Automator keywords and actions are predefined and implemented

10 1 Marks
In the process of test development, the Automator uses their expertise to map the application components to be tested with the automation tool(s) that will execute the tests.

Answer: ○True ○False

Lesson

2 Installing RRAFS

SAFS for Rational Robot

OK, enough of the touchy-feely stuff. We want to run somethin'!
Installing RRAFS is actually very simple and trouble-free *if* we take heed to the installation prerequisites and don't assume we can just run the install script.

In this lesson we will review those prerequisites and make sure the system is ready for the install. Then we will download and install the latest version. We'll finish off with information on uninstalling RRAFS.

Installing RRAFS and the SAFS Framework

Quiz: Installing RRAFS and the SAFS Framework

Installing RRAFS and the SAFS Framework

RRAFS Installation Prerequisites

RRAFS requires:

1. **Windows Scripting Host 5.6**
   You can verify this version of WSH is installed on your system by typing the following at any command prompt and pressing ENTER:

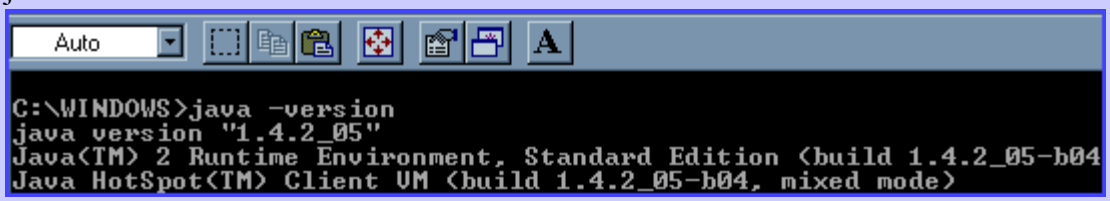
   cscript

   If the command does not correctly execute, or if the version is less than 5.6, you will need to install version 5.6 or later from the Microsoft Scripting Downloads website.

2. **Java Runtime 1.4** (for the SAFS Framework and STAF)
   You can verify this version of Java is installed on your system by typing the following at any command prompt and pressing ENTER:

   java -version

   If the command does not correctly execute, or if the version is less than 1.4, you will need to install version 1.4 or later from the Java 2 Platform website.

While the RRAFS install script *Setup.VBS* will verify the correct versions are present, it will not install the correct versions if they are not. Thus, it may be a good idea to make sure prior to running the installation.

Microsoft Scripting Downloads

http://msdn.microsoft.com/downloads/list/webdev.asp

Java 2 Platform

http://java.sun.com/j2se/

RRAFS for IBM Rational Robot

There is a minor distinction that must be made concerning RRAFS if it hasn't already sunk in.

RRAFS is for Rational Robot.

RRAFS can only be installed *AFTER* IBM Rational Robot has been installed. This is because RRAFS must be installed into the *<root>\Rational\Rational Test\sqabas32* directory where the Rational Software has been installed.

RRAFS can leverage XDE Tester through the formidable power of the SAFS Framework, but RRAFS can also work just fine without SAFS. However, if you intend to leverage the wider array of test tools provided by the SAFS Framework then you must install SAFS.

SAFS for IBM Rational XDE Tester

There is a minor distinction that must be made concerning RRAFS and SAFS if it hasn't already sunk in.

RRAFS provides for Rational Robot.
SAFS provides for Rational XDE Tester.

As mentioned, RRAFS can only be installed *AFTER* IBM Rational Robot has been installed. However, SAFS can be installed at any time -- even if neither Robot nor XDE Tester are installed. SAFS can be installed at some future time after RRAFS if that is desirable. But it is easiest if SAFS is installed automatically as part of the RRAFS installation.

As mentioned, support for IBM Rational XDE Tester is provided by the SAFS Framework. Officially, the SAFS Engine for XDE Tester is called SAFS/RobotJ and sometimes RJ, for short.

RRAFS is not required to take advantage of SAFS/RobotJ, but RRAFS provides the most full-featured Driver for SAFS/RobotJ at this time. Unfortunately, it is beyond the scope of this tutorial to go into detail of how to use the SAFS/RobotJ engine without RRAFS. Keep your eyes peeled for future tutorials or enhancements on this subject!

Installing RRAFS and the SAFS Framework

Since you've already gone through the prerequisites (you have gone through all the prerequisites, right?) we will now commence the excitement of installing the RRAFS and SAFS Frameworks!

REFERENCES:
RRAFS FAQS
How Do I...?
Quick Reference
Keyword Reference
Old-Style Reference
RRAFS Libraries
RRAFS Install/Setup

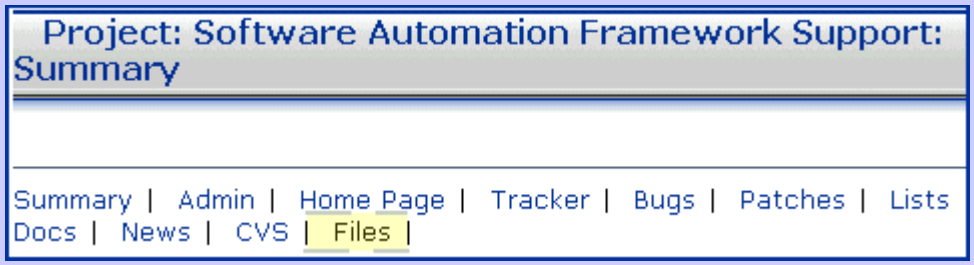
SUPPORT LISTS

DOWNLOADS

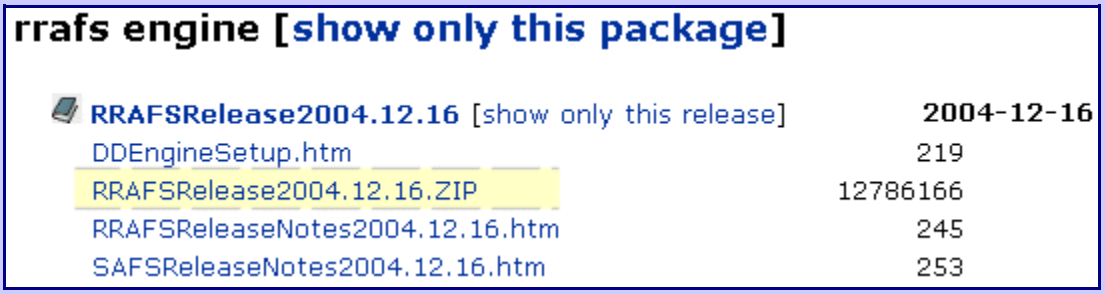
DEVELOPER LINKS

You can get to the RRAFS Framework release thru the "**DOWNLOADS**" link highlighted on the left on the SAFS Home Page at http://safsdev.sourceforge.net.

You can also get to the framework release thru the "**Files**" link highlighted below on the SAFS Project Page on SourceForge at http://sourceforge.net/projects/safsdev.



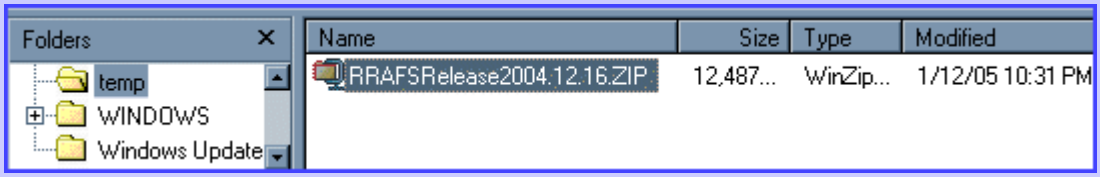
Locate the latest *Release* from the "RRAFS Engine" section of the Files page. (*Patches*, if present, can only be applied upon a previously installed release.) Review the release and notes or setup docs and then download the release **ZIP** file into a temporary location of your choice. The DDE_RUNTIME location (*.\Rational\Rational Test\sqabas32*) where Rational Software is installed is also an excellent location to store the ZIP file. For this tutorial we'll assume C:\TEMP.



RRAFS Engine

http://sourceforge.net/project/showfiles.php?group_id=56751&package_id=53358

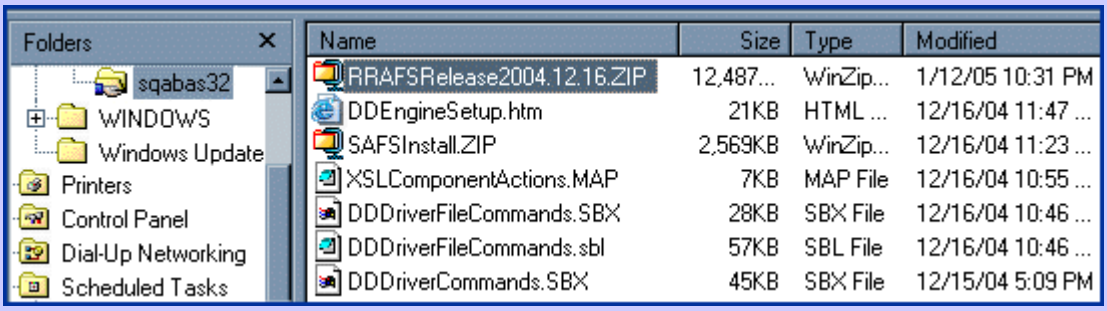
Extract the RRAFS Release into DDE_RUNTIME



Assuming the downloaded RRAFS Release was placed in C:\TEMP, we must now EXTRACT the contents of the ZIP file into the DDE_RUNTIME directory where Rational Software has been installed. The path to the specific subdirectory of the Rational Software install that we call DDE_RUNTIME is:

DDE_RUNTIME: .\Rational\Rational Test\sqabas32

Note: All files must be extracted into this specific directory. It is not optional.



Standard RRAFS Install

Setup.VBS is the RRAFS Install script and will be located among the files extracted into the DDE_RUNTIME directory. There are no command-line options for the Setup.VBS script. If you like, you can open the Setup.VBS script in your favorite text editor and view some of the commented details of what the script does.

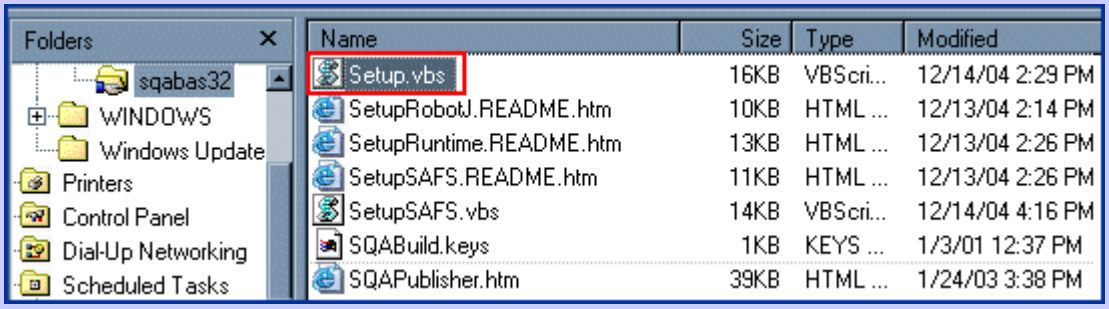
A full RRAFS install uses the following installation scripts:

1. Setup.VBS main installation script.
2. SetupSAFS.VBS invoked to install the SAFS Framework
3. Java "*org.safs.installer.SilentInstaller*" class

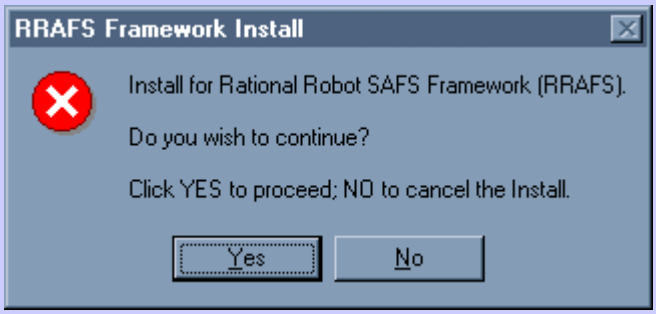
The Java *SilentInstaller* class is invoked by *SetupSAFS.VBS* only if the user does not Cancel the request to install the SAFS Framework. The SAFS Framework does not have to be installed with RRAFS and the user will be prompted to approve the installation. The SAFS Framework install can be done separately and this separate installation is covered in the SAFS101 Introduction to SAFS Tutorial.

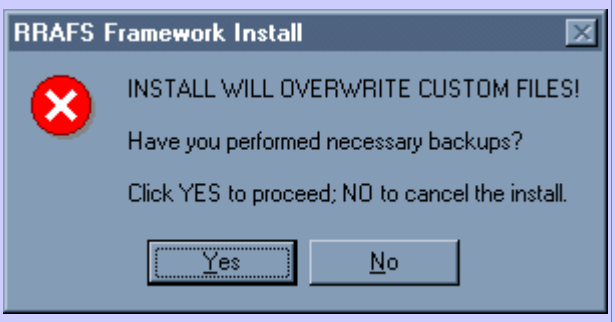
OK...so let's go ahead and finish the installation...

Execute the RRAFS Install Script: Setup.VBS



The Standard Install is not "pretty". But it does automate some otherwise complex tasks for the user. Simply **double-click** the **Setup.vbs** script to launch the installer.



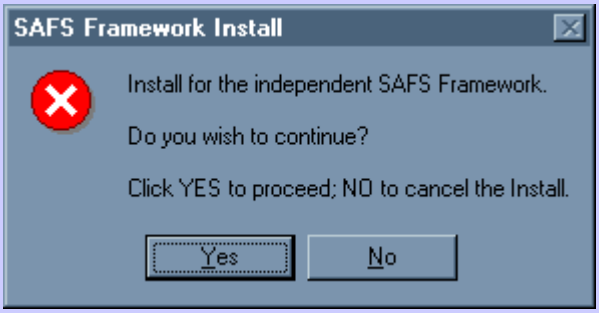


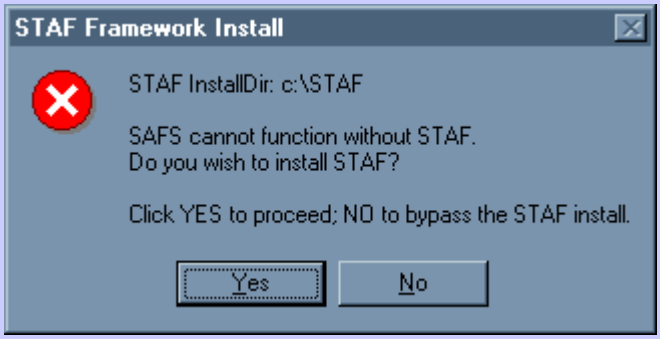
The above installer dialogs require a user response. Other dialogs may appear during the install, but they simply provide status information. They will display for a few seconds and the installation will continue.

Upon finishing the main installation, the RRAFS installer will automatically invoke the SetupSAFS.vbs installer. We recommend you install SAFS along with RRAFS even if you initially do not intend to take advantage of the new tools and commands it adds.

Continue with the SAFS Framework Install

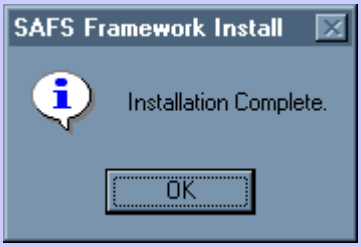
After the main RRAFS install the SAFS Framework install will be launched...





The above installer dialogs require a user response. Other dialogs may appear during the install, but they simply provide status information. They will display for a few seconds and the installation will continue.

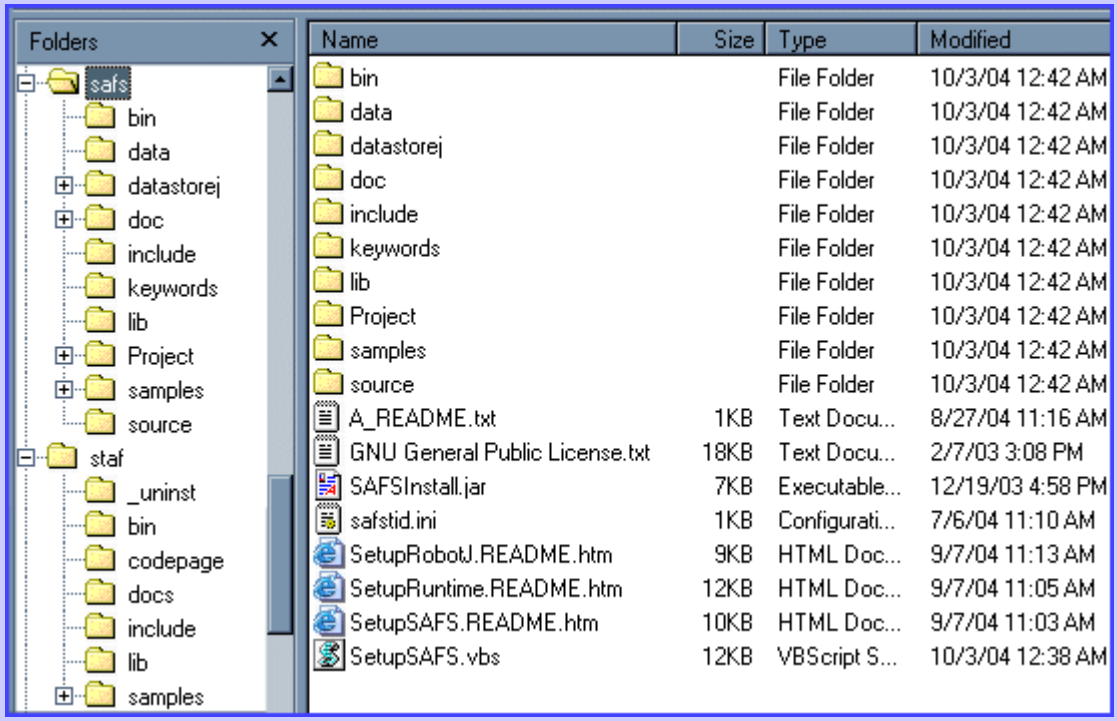
After a silent Java install of STAF lasting a minute or so, the installation will display the completion dialog shown below.



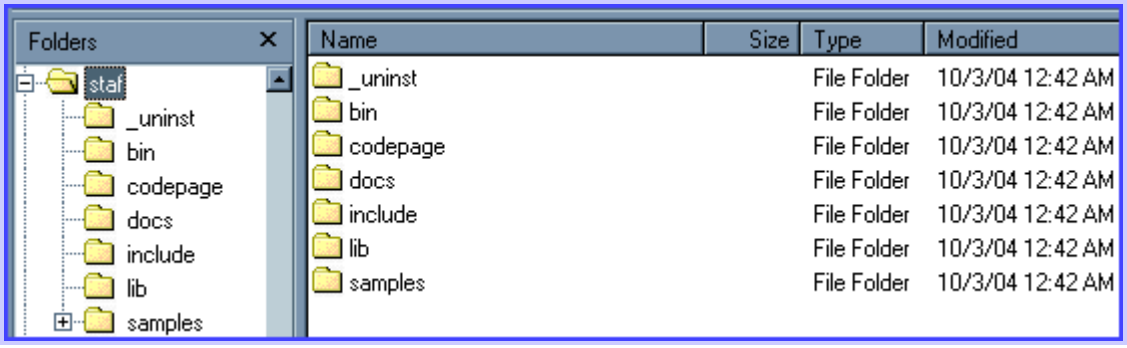
And there you go! The RRAFS Installation is complete!

Verify a successful SAFS Framework Install

Following a full successful RRAFS install, you should see similar files and directories below in the root directory where SAFS was installed. (By default this will be C:\SAFS.)



If the STAF install was not bypassed, you should see similar directories below in the root directory where STAF was installed. (By default this will be C:\STAF.)

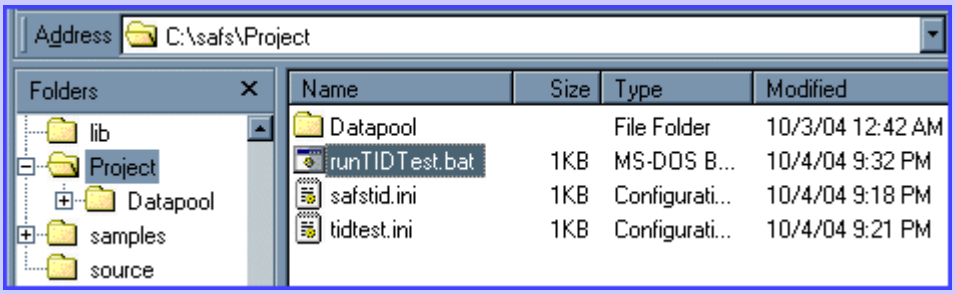


Execute a Simple SAFS Test

You can also verify the installation of all SAFS components by executing a simple test installed with the SAFS Framework. If you bypassed the SAFS install then this is pretty unlikely to work!

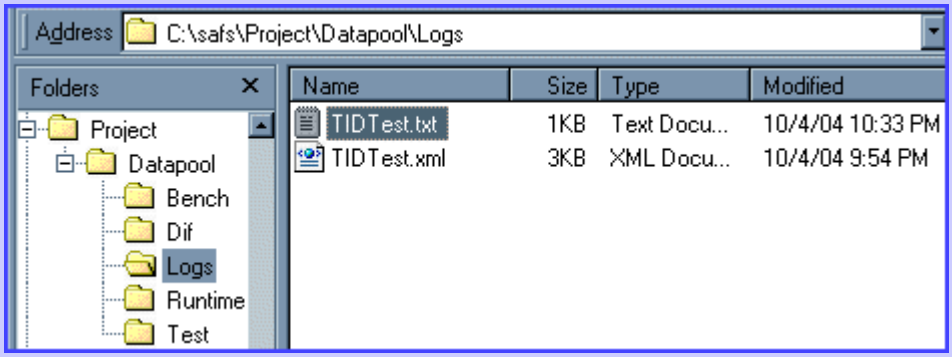
The "runTIDTest.bat" file shown below is preset to execute if you installed SAFS in the default C:\SAFS directory. If you installed to custom directories you will need to edit the path information in "runtTIDTest.BAT", "TIDTest.INI", and "SAFSTID.INI".

To execute the test, double-click "runTIDTest.bat"



With everything working as planned, you should see a CMD window appear and the test execute. The CMD window will display information similar to what you can view here.

Upon test completion, you should have a Text log and an XML log in the Project\Datapool\Logs directory as shown below:



Open that TIDTest.TXT log file and you should find a test log very similar to the one you can view here. There is still some Rational Robot configuration to do in the separate lesson on Enabling Rational Robot, but this completes the actual installation steps for the RRAFS and SAFS

CMD Window Display

http://safsworks.com/online/image/tidout.txt

TIDTest.TXT Log

http://safsworks.com/online/image/TIDTest.txt

Uninstalling RRAFS, SAFS, and STAF

Uninstalling the various framework pieces is not complicated, but it is generally not automated either. STAF has an Uninstaller.exe that will remove STAF with some coaxing, but RRAFS and SAFS are manual removal processes.

Before we begin, some notes on ifs, whys and whatfors:

1. Obviously, if you did not install the SAFS Framework, or did not let RRAFS install the SAFS Framework, then you don't have to uninstall SAFS and STAF because they will not be present.
2. You can uninstall RRAFS and the SAFS Framework yet retain STAF and keep it installed. STAF is a very useful framework on its own. Check out the STAF Homepage and see for yourself.
3. You cannot uninstall STAF and expect the SAFS Framework to function. The SAFS Framework requires STAF. However RRAFS can still function without a functional SAFS Framework.

Let's begin with the very simple process of uninstalling STAF. You can always skip that one if you wish to keep STAF installed...

STAF Homepage

http://staf.sourceforge.net

Uninstalling STAF

Removing STAF is pretty simple because it actually has an Uninstaller.exe! However, the automated installation of STAF by RRAFS and the SAFS Framework uses the silent install option for STAF. Thus, STAF will not recognize its typical uninstall process. We must perform a silent uninstall!

The default STAF Installation directory is: **C:\STAF**
If you installed STAF do a different directory you must modify the following path information appropriately.

From any command prompt you would execute the following command:

C:\STAF_uninst\uninstaller.exe -silent

The uninstall will take several moments and since it is a silent uninstall you will not receive any dialog box or prompt that it has completed. Sorry, but that is just the way it is.

If you are so inclined, you can also delete all references to STAF JAR files in the System CLASSPATH Environment variable. The CLASSPATH environment variable can generally be edited by:

1. **Right-Click the *My Computer* icon on the Desktop**
2. **Select *Properties* from the popup menu**
3. **Select the *Advanced* tab in the dialog**
4. **Locate and Edit the CLASSPATH Environment Variable**

Remove all JAR file references that contain paths to STAF Installation directories. You may also wish to delete the STAFDIR environment variable.

If you have already uninstalled the SAFS Framework, you can check and make sure all SAFS JAR references have also been deleted from the CLASSPATH. You may also wish to delete the SAFSDIR environment variable.

Uninstalling the SAFS Framework

Removing the SAFS Framework is practically effortless!

Locate and delete all files in the SAFS installation directory and all subdirectories. The default SAFS Installation directory is: **C:\SAFS**

If you are so inclined, you can also delete all references to SAFS JAR files in the System CLASSPATH Environment variable. The CLASSPATH environment variable can generally be edited by:

1. **Right-Click the *My Computer* icon on the Desktop**
2. **Select *Properties* from the popup menu**
3. **Select the *Advanced* tab in the dialog**
4. **Locate and Edit the CLASSPATH Environment Variable**

Remove all JAR file references that contain paths to the SAFS Installation SAFS\lib directory that were just deleted. You may also wish to delete the SAFSDIR environment variable.

If you have already uninstalled STAF, you can check and make sure all STAF references have also been deleted from the CLASSPATH. You may also wish to delete the STAFDIR environment variable.

Uninstalling RRAFS

Removing RRAFS is only slightly more complicated than removing the SAFS Framework. A few files installed with RRAFS were copied to the *<WINROOT>\System32* directory and the DDVariableStore.DLL was registered with Windows. These things will need to be undone.


Locate and delete the following files in the **<WINROOT>\System32** directory. This is where the bulk of the Windows operating system files are located. Be careful, and don't be deleting anything beyond what is listed here.

CWPHooker.DLL
STAFWrap.DLL
MSXSL.EXE (optional -- not RRAFS-specific)

The rest of the uninstall process involves files in the DDE_RUNTIME directory. Remember, DDE_RUNTIME is the *.\Rational Test\sqabas32* subdirectory off the Rational Software installation directory. The default path is below. If your installation is different your path must be adjusted accordingly.

DDE_RUNTIME = (default) **C:\Program Files\Rational\Rational Test\sqabas32**

Before you can delete the files in the DDE_RUNTIME directory, you need to unregister the DDVariableStore.DLL file that resides there.

At the **Start->Run** command prompt unregister the DLL with the following command. Of course, <DDE_RUNTIME> must be replaced with the correct path and any path containing spaces must be enclosed in quotes as shown below.

regsvr32 -u "<DDE_RUNTIME>\DDVariableStore.DLL"


Finally, we can delete the RRAFS files in our DDE_RUNTIME directory. *However, DO NOT delete the files that are actually part of the Rational Software install!* The files that should remain are listed below. Delete everything else.

dpconst.sbh **global.*** (may not be present)
sqautils.sbh **sqautils.sbl** **sqautils.sbx**

There you go! RRAFS is now uninstalled.

Quiz: Installing RRAFS and the SAFS Framework

1 1 Marks — What is the minimum version of the Windows Scripting Host required for RRAFS?

Answer:
- (●) a. WSH Version 5.6 or later
- ( ) b. WSH Version 3.0 or later
- ( ) c. WSH Version 5.1 or later
- ( ) d. WSH Version 1.0 or later

2 1 Marks — What is the minimum required version of Java for the SAFS Framework and, consequently, RRAFS?

Answer:
- ( ) a. Java 1.2 or later
- ( ) b. Java 1.1 or later
- (●) c. Java 1.4 or later
- ( ) d. Java 1.3 or later

3 1 Marks — Should we install RRAFS before Rational Robot?

Answer:
- ( ) a. It doesn't matter.
- ( ) b. Yes. RRAFS should definitely be installed before Robot.
- (●) c. No. RRAFS should be installed AFTER Robot

4 1 Marks — The SAFS Framework (and STAF) can be installed at anytime before, during, and even after other tools like RRAFS are installed.

Answer: (●) True ( ) False

5 1 Marks — RRAFS is a SAFS Engine for which specific automation tool?

Answer:
- ( ) a. Mercury Interactive WinRunner
- ( ) b. IBM Rational XDE Tester
- ( ) c. IBM Rational Functional Tester
- (●) d. IBM Rational Robot

6 1 Marks — The SAFS/RobotJ engine for IBM Rational RobotJ and XDE Tester is considered part of the RRAFS Framework.

Answer: (●) True ( ) False

7 1 Marks — The downloaded RRAFS Release MUST be UnZipped or Extracted into which directory?

Answer:
- (●) a. DDE_RUNTIME: .\Rational\Rational Test\sqabas32
- ( ) b. Any temp directory
- ( ) c. C:\STAF
- ( ) d. C:\SAFS

8 1 Marks
Once the RRAFS Release is extracted into the proper directory, what is the name of the setup script for RRAFS?

Answer:
- a. DDVariableStoreREGISTER.vbs
- b. Setup.VBS
- c. MSXSL.EXE
- d. SetupSAFS.vbs

9 1 Marks
The SAFS Framework can be installed independent of RRAFS. What is the name of the setup script used to perform a Wizard-like install of the SAFS Framework?

Answer:
- a. DDVariableStoreREGISTER.vbs
- b. MSXSL.EXE
- c. SetupSAFS.vbs
- d. Setup.VBS

10 1 Marks
What is the default installation directory for the independent SAFS Framework on Windows?

Answer:
- a. C:\SAFS
- b. DDE_RUNTIME: .\Rational\Rational Test\sqabas32
- c. C:\STAF
- d. C:\Program Files\SAFS

11 1 Marks
The SAFS Framework requires and bundles STAF. What is the default installation directory for STAF?

Answer:
- a. C:\Program Files\STAF
- b. C:\STAF
- c. C:\SAFS
- d. DDE_RUNTIME: .\Rational\Rational Test\sqabas32

12 1 Marks
After a successful install of the SAFS Framework there is a quick test that verifies the SAFS installation seems good. What is the name of the script used to execute this test?

Answer:
- a. runTIDTest.rec
- b. SAFSTID.INI
- c. runTIDTest.bat

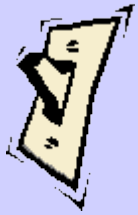
13 1 Marks
When removing RRAFS from its installation directory, what 4 files installed by Rational must NEVER be deleted?

Answer:
- [ ] a. sqautil.sbh
- [ ] b. sqautil.sbx
- [ ] c. global.sbx
- [ ] d. sqautil.sbl
- [ ] e. dpconst.sbh

This Page Intentionally Blank

Lesson

3



Enabling Rational Robot

After we have installed RRAFS, there are some optimizations needed to allow Robot to run at peak performance. Some of these are optional, but most are highly recommended.

While we're in there we will explore how a Rational project repository is configured for RRAFS testing and what the RRAFS.INI configuration file provides.

Enable Rational Robot

Quiz: Enabling Rational Robot

Enable Rational Robot

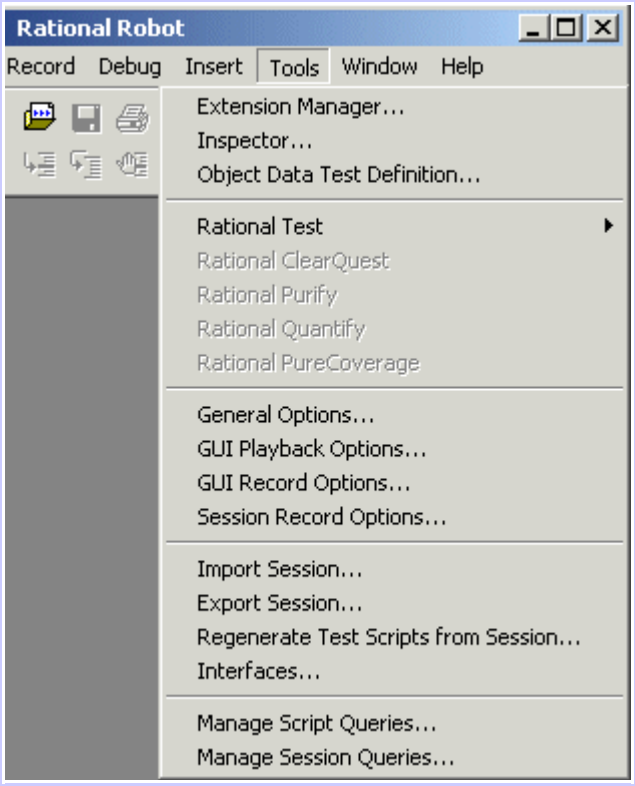
Enabling IBM Rational Robot

Here we are going to tinker with some of the Rational Robot Tools menu options to optimize Robot for RRAFS.

We will be editing important Playback settings. Then we will strongly encourage you to create and then enable a separate RRAFS101 project repository for future topics and lessons.

Finally we will review an existing sample test and the RRAFS.INI file before we jump in and muck around with them!

Review\Edit Rational Robot Playback Settings

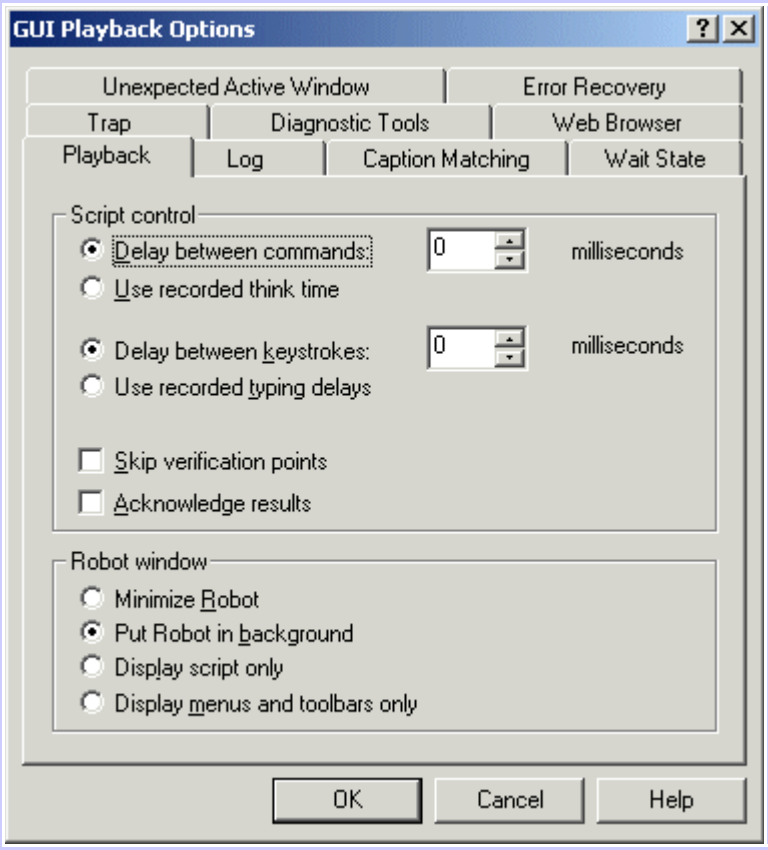


Having installed RRAFS, we need to go into Robot and perform some setup that enables it to perform at its best. These setup steps will be performed by accessing the Tools menu shown to the left. We will be looking primarily at the **Extension Manager** and **GUI Playback Options**.

Remember, RRAFS is not a collection of scripts. We are effectively inserting a new test language interpreter on top of Robot. For this reason, the default settings used by Robot to run traditional scripts are not ideal for something as complex as RRAFS. For example, we don't want Robot to abort execution when an error occurs. We don't want Robot to attempt to dismiss unexpected windows. We want Robot to continue executing this new framework and let the framework deal with issues that arise.

If you have not already done so, launch Rational Robot and log on to any Rational project repository so that we can make these changes. If you do not have any existing Rational project repository then go back to the "Enable Rational Robot" lesson branch page and select "Enable Project Repository". Remember to return to this lesson branch once you have created your new project repository.

Set Robot Playback Delays

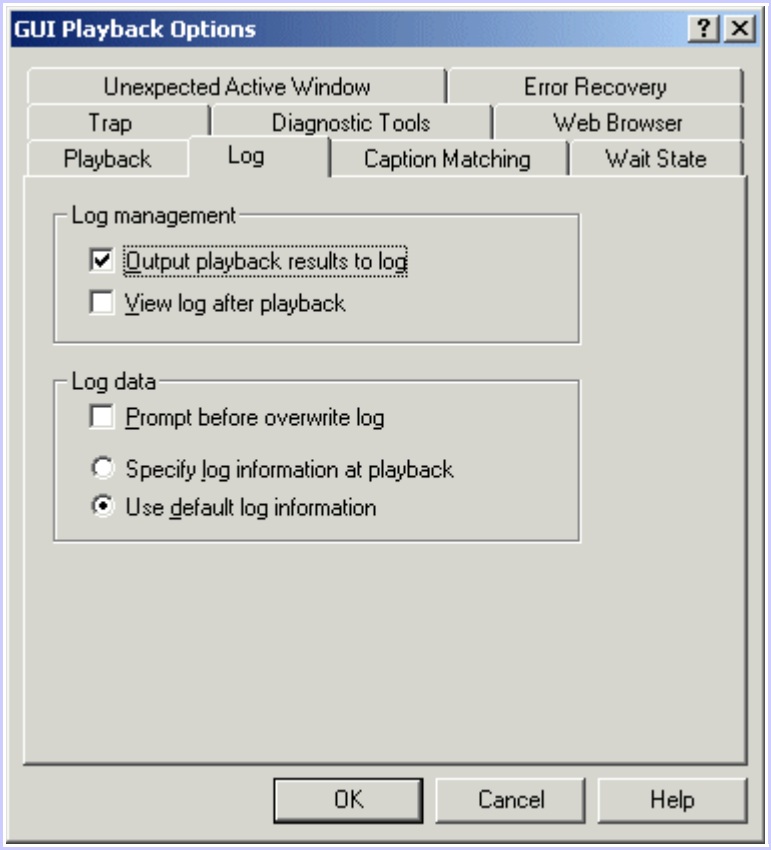


Make the following changes to the **GUI Playback Options** available through Robot's Tools Menu. These will be found on the **Playback** tab:

Delays Between Commands: 0ms
Delays Between Keystrokes: 0ms

Set your preference in the **Robot Window** panel for how you want Robot to appear during playback. Keep Robot in the background if you want to be able to see the Console during playback.

Set Robot Playback Log Options



Verify the following Log Management settings in the **GUI Playback Options** available through Robot's Tools Menu. These will be found on the **Log** tab:

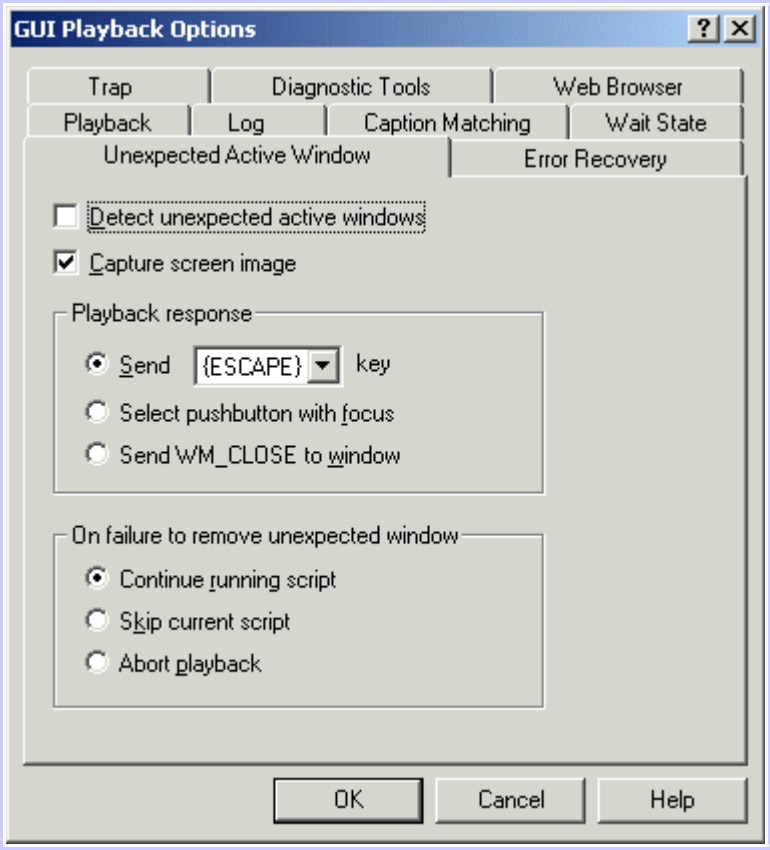
Select:
Output playback results to log

This must be enabled even if we don't care to use Robot's log. Some VPs and tools that RRAFS will use dynamically at runtime will not function properly if the TestManager log has been completely disabled.

View log after playback should be set according to the user's preference. RRAFS performs its own logging and a tester can often get by without ever seeing the Robot log.

Log data settings are also set per the user's preference.

Disable Unexpected Active Window Options



Make the following changes to the **GUI Playback Options** available through Robot's Tools Menu. These will be found on the **Unexpected Active Window** tab:

Unselect**:**
Detect unexpected active windows

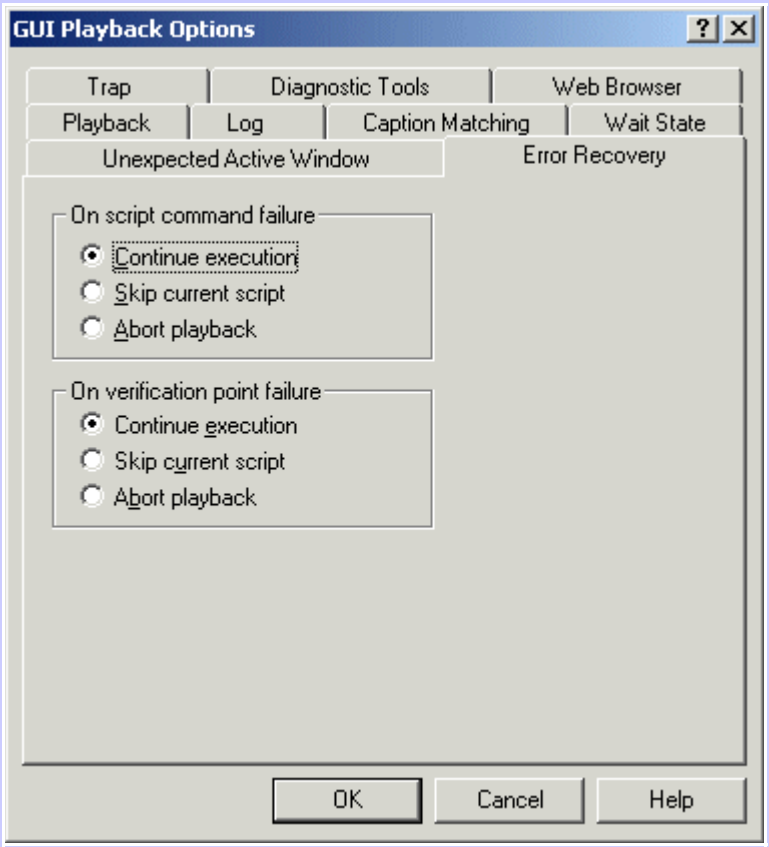
Capture a screen image will be a user preference, but this might not matter once the detection mechanism is disabled.

The **Playback response** and **On failure to remove unexpected window** settings should (hopefully) be inconsequential once we disable the detection of unexpected windows. But just in case...

The safest **Playback response** would be to **Send {Escape} key**.

On failure to remove unexpected active window should be set to **Continue running script** just to be safe. We really don't want anything to stop the RRAFS framework code right in the middle of execution.

Set Error Recovery Options



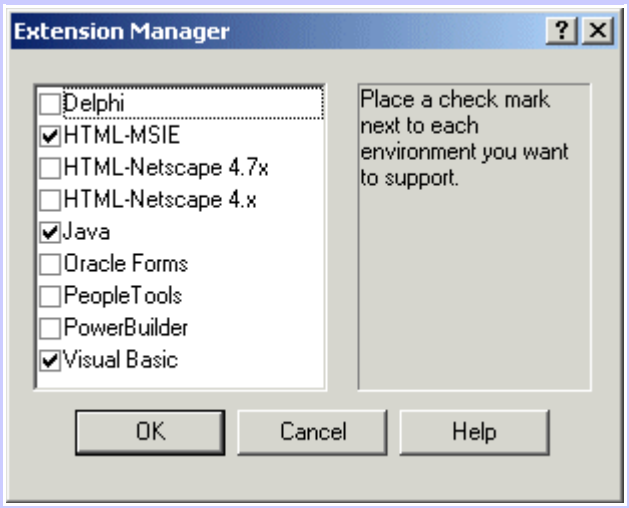
Make the following changes to the **GUI Playback Options** available through Robot's Tools Menu. These will be found on the **Error Recovery** tab:

Continue execution

should be selected for both **On script command failure** and **On verification point failure.** We want to keep on keepin' on in all cases.

Once all the changes have been, press the **OK** button to save everything and close the dialog.

Disable Unnecessary Extensions



Disable unnecessary environments in the **Extension Manager** available through Robot's Tools Menu.

This is very important. Performance can be severely reduced if extensions for application environments we will not be testing are left enabled.

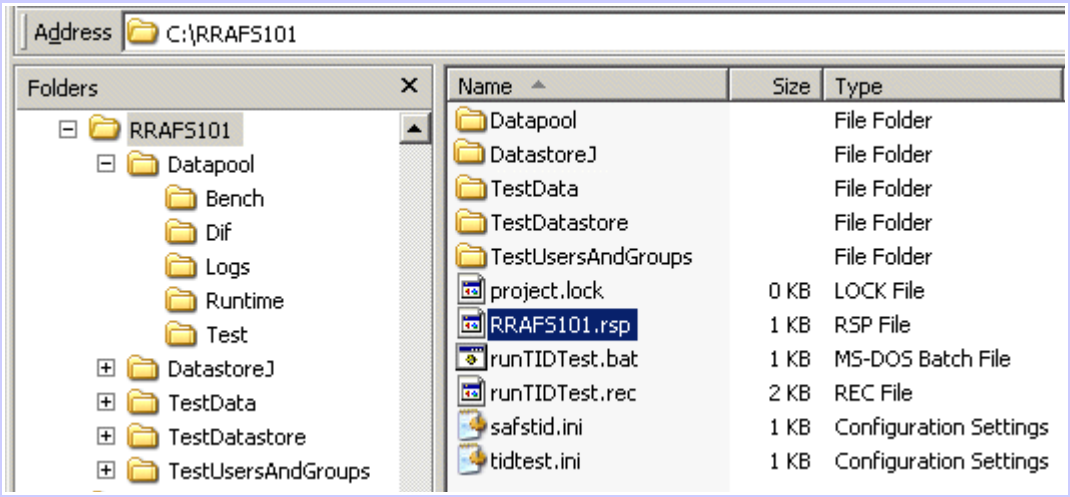
When Robot searches for components in our application, it searches every enabled environment until it finds the component. By reducing the number of environments searched Robot will find our application objects faster.

Press **OK** to save your changes once you are finished. Robot will probably tell you these settings will not take effect until you close and restart Robot.

Well, we are done for now. So go ahead and close Robot, take a short break, and be happy!

Create\Enable a RRAFS101 Project Repository

This is one place where we instructors get to sluff off just a little bit.
For the purposes of the rest of this tutorial we will be referring to a RRAFS101 Project Repository. We are going to assume this RRAFS101 Project Repository was created in the **C:\RRAFS101** directory. All our screenshots and discussions will reflect this.
(Below is an example of a fully RRAFS-enabled Project Repository.)

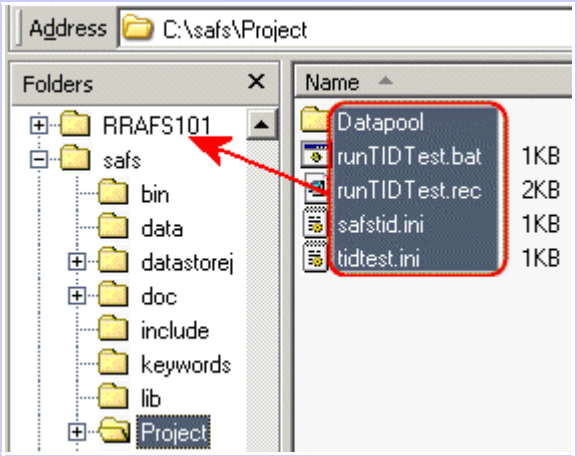


So we instructors are going to sit here and wait while you go into the Rational Administrator and create this new pristine Project Repository. There are too many "where froms" and "which ones" for us to deal with in a remote tutorial, and this is a proficiency you should have already.

Now, you could just use an existing Project Repository if you dare. But then you have to keep track of what we mean when we mention the RRAFS101 project and project directories. That is up to you.

When you are ready, we will proceed with the additional setup necessary to fully RRAFS-enable the Project Repository.

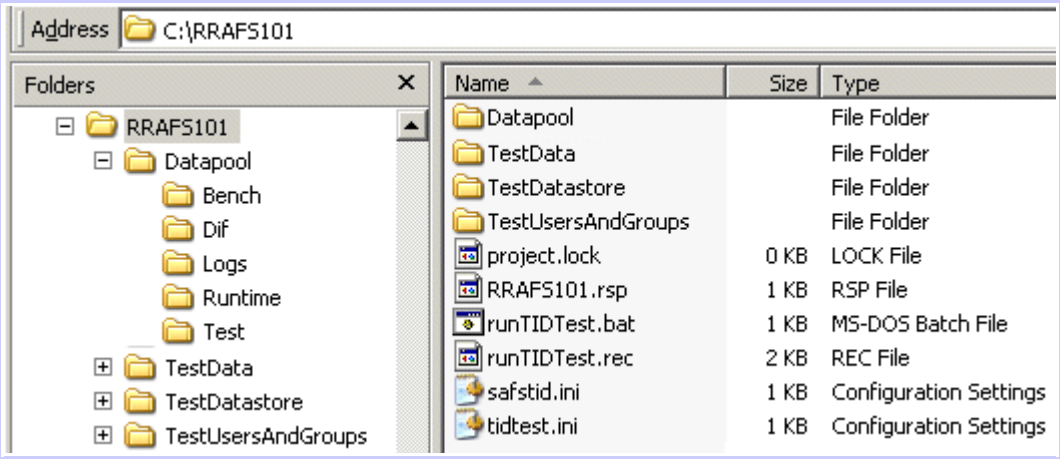
Copy SAFS\Project into RRAFS101 Project



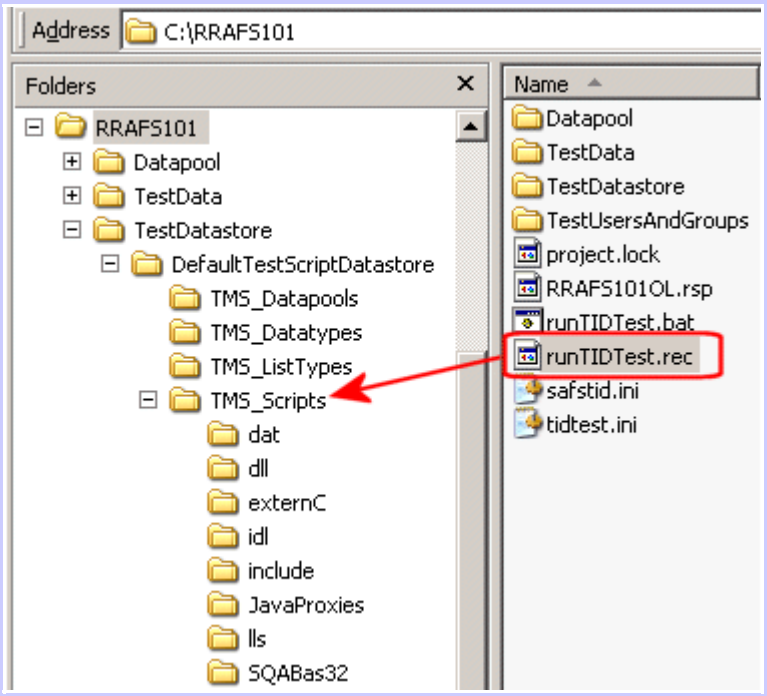
Welcome back! We're going to trust you on this one.

We now have a brand new Rational repository in C:\RRAFS101, right?! Good. Now let's add our SAFS project stuff to it.

Copy the contents of the **SAFS\Project** folder, including files and subfolders, but not the "Project" folder itself into the root directory of the repository. In this case: **C:\RRAFS101**. You should end up with a RRAFS101 directory like below:



Copy runTIDTest.rec Script

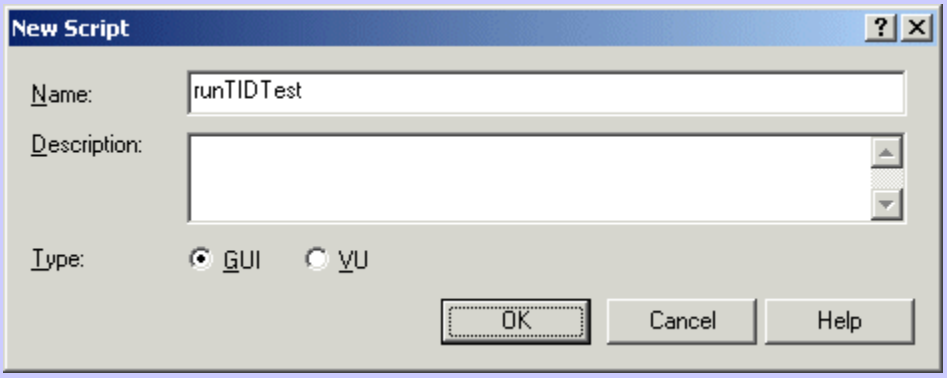


We are now going to copy the Robot **runTIDTest.rec** script to the TMS_Scripts directory. This is the directory within each Rational repository where all scripts are stored.

Go ahead and copy the **runTIDTest.rec** file from our **C:\RRAFS101** directory into the **TMS_Scripts** directory as shown at left.

Launch Robot and login to this RRAFS101 project and import this script with a **File->New Script Ctrl+N** operation from Robot's File Menu. Use "runTIDTest" as the name of the script. The script name must exactly match the root name of the script we just copied as shown below.

Press **OK** to finish the import and the complete script like the one on the next page should appear in Robot's Editor.



Sample runTIDTest.rec:

```
OPTION EXPLICIT

'$INCLUDE: "DDEngine.sbh"

Sub Main

    Dim status As Integer
    Dim logfile As String
    Dim summary as String
    Dim logsdir as String
    Dim datadir as String

    SetDDEDatapoolDirectory(Environ("SAFSDIR") &"\project\datapool")
    DDGClearAppMapCache
    logsdir = GetDDELogsDirectory()
    datadir = GetDDEDatapoolDirectory()

    On Error Resume Next

        MainLog.logid = "TIDTest.Robot"
        logfile = logsdir & MainLog.logid
        summary = logfile &".htm"

        kill logfile & ".txt"
        DoEvents

        InitLogFacility TEXTLOG_ENABLED OR CONSOLE_ENABLED, MainLog
        DoEvents

    On Error Goto 0

    CDCycleDriver "TIDTest.cdd", ",", MainLog, MainLog, MainLog, CDCycleDrivenMode

    LogMessage " ", MainLog

    'close the text log
    CloseAllLogs MainLog, 1
    Reset

    DoEvents

    SAFSShutdownDriverCommands
    SAFSShutdownRobotJ
    DelayFor 2000
    SAFSShutdownSTAF

    StartApplication "notepad.exe "& MainLog.textlog

End Sub
```

Compile runTIDTest.rec Script

Before we compile this script we want to point out something about it.

The first executable line of the script (shown below) alters where the test input will come from and where the output log will be written.

SetDDEDatapoolDirectory(Environ("SAFSDIR") &"\project\datapool")

With the above line in place the test is going to read data from the **SAFS\Project\Datapool** directory and write its log to the **SAFS\Project\Datapool\Logs** directory.

If you remove or comment out that line then it will use the default Project directories -- in this case: **C:\RRAFS101\Datapool** and **C:\RRAFS101\Datapool\Logs**. Since we now have a real viable RRAFS101 project in place it is recommended you delete this line in the script.

Go ahead and save the changes and compile the script. Hopefully, there are no surprises during this short compile! If there are...well...you are going to have to use your expertise and fix 'em.

Review Contents of runTIDTest Robot Script (Part 1)

In the previous section we imported the the runTIDTest.rec script into our RRAFS101 project and compiled it with Rational Robot. Let's review what is actually in this very short, simple script. An online copy can be found here.

'$INCLUDE: "DDEngine.SBH"
This is the King of RRAFS SQABasic library headers. With this one statement SQABasic scripts gain access to virtually every important Public Function, CONSTANT, and User-Defined Type coded for RRAFS.

DDGClearAppMapCache
(http://safsdev.sourceforge.net/sqabasic2000/DDGUIUtilities.htm#ddgclearappmapcache)
This important function resets internal and external App Map data that may have been stored during previous runs. This is especially important if you are editing App Maps between test runs or while debugging.

GetDDELogsDirectory and GetDDEDatapoolDirectory
(http://safsdev.sourceforge.net/sqabasic2000/DDUtilities.htm#getddelogsdirectory)
(http://safsdev.sourceforge.net/sqabasic2000/DDUtilities.htm#getddedatapooldirectory)
Forces the framework to evaluate any RRAFS.INI directory settings and returns the path to the directories that will be used -- at least initially -- for the test.

MainLog and InitLogFacility
(http://safsdev.sourceforge.net/sqabasic2000/LogUtilities.htm#user_defined)
(http://safsdev.sourceforge.net/sqabasic2000/LogUtilities.htm#initlogfacility)
MainLog is the default RRAFS LogFacility. A LogFacility holds information of RRAFS logs we will use during the test. MainLog is generally the only LogFacility needed for the test, but more than one LogFacility can be initialized and used for some odd reasons. Note the KILL statement that deletes any previous log before we initialize the new one with InitLogFacility.

CDCycleDriver
(http://safsdev.sourceforge.net/sqabasic2000/CycleDriver.htm#cdcycledriver)
Once logging has been initialized we launch the test! This shows launching a Cycle level test, but scripts can commence testing at the SuiteDriver or StepDriver level, too. Note the use of the one MainLog for all possible LogFacilities allowed by this routine. This writes all log messages to one log for all test levels instead of writing to separate logs for each test level.

Review Contents of runTIDTest Robot Script (Part 2)

Continuing with our review of the runTIDTest.rec script (viewable here):

LogMessage
(http://safsdev.sourceforge.net/sqabasic2000/LogUtilities.htm#logmessage)
Just writing a blank spacer line to the MainLog log(s) from within this script. Scripts and the keyword-driven engine launched with CDCycleDriver all write to the same logs. When SAFS is enabled in later lessons Robot and all tools external to Robot will write to the same SAFS logs. But all of this will still occur through LogMessage.

CloseAllLogs
(http://safsdev.sourceforge.net/sqabasic2000/LogUtilities.htm#closealllogs)
Close all the log(s) initialized in the MainLog LogFacility. This script only has the TEXTLOG_ENABLED, but XML logging could also be enabled. This one call will make sure that all logs opened in the LogFacility are closed. This closing of logs only applies to the RRAFS logs we physically write to files. Currently this is only text logs and XML logs.

SAFSShutdownDriverCommands, SAFSShutdownRobotJ, SAFSShutdownSTAF
(http://safsdev.sourceforge.net/sqabasic2000/SAFSUtilities.htm#safsshutdowndrivercommands)
(http://safsdev.sourceforge.net/sqabasic2000/SAFSUtilities.htm#safsshutdownrobotj)
(http://safsdev.sourceforge.net/sqabasic2000/SAFSUtilities.htm#safsshutdownstaf)
Although right now we have not enabled the external SAFS framework, this script is ready for it. Calling these shutdown functions when the SAFS framework is not being used is just fine.

Lastly the SQABasic StartApplication call launches Notepad so that we can view the test log when the script is finished.

Review Contents of the TIDTest.CDD Test Table

When we copied the SAFS\Project contents into the RRAFS101 Project we also copied the simple test that the runTIDTest script attempts to execute -- **TIDTest.CDD**. This is found in the **RRAFS101\Datapool** directory or here. (The Datapool directory is the default location for ALL test tables in a project.)

Note that this test table uses a comma field separator.

This test is a very simple test. All but one of the test records are Driver Commands (**C**). The BlockID (**B**) test record is nothing more than a named label identifying that line in the test table. You will notice that its label "*ERROR*" is the target for the *OnNotEqualGotoBlockID* Driver command above it.

You can review the official SAFS Record Formats for more info on the format of test records if you would like.

The SAFS Keyword Reference is the detailed reference for all SAFS keywords.
The SAFS Quick Reference provides for quick lookup and links to details.

Below are direct links to the 6 Driver Commands used in this simple test:

SetApplicationMap	OnNotEqualGotoBlockID	LogTestSuccess
Concatenate	ExitTable	LogTestFailure

Notice that the very first Driver Command "*SetApplicationMap*" has specified the **TIDTest.MAP** Application Map. We will review that next.

Sample TIDTest.CDD:

```
C, SetApplicationMap, TIDTest.MAP
C, Concatenate, ^var1 , ^var2, "result"

C, OnNotEqualGotoBlockID, "ERROR", ^result, "abcdef"

C, LogTestSuccess, "TIDTest Services all present and working."
C, ExitTable

B, ERROR
C, LogTestFailure, "Error in one or more SAFS modules!"
```

SAFS Record Formats
http://sourceforge.net/docman/display_doc.php?docid=17266&group_id=56751

SAFS Keyword Reference
http://safsdev.sourceforge.net/sqabasic2000/RRAFSReference.htm

SAFS Quick Reference
http://safsdev.sourceforge.net/sqabasic2000/RRAFSQuickReference.htm

Review Contents of TIDTest App Map

So let's review the Application Map **TIDTest.MAP**. This is found in the **RRAFS101\Datapool** directory or here.

Note that an Application Map is in a standard (Windows) INI file format -- Name\Value pairs grouped into named Sections. There is but one reserved section ID for SAFS Application Maps and that is **[ApplicationConstants]**. And that also happens to be the only section defined in this TIDTest.MAP file.

ApplicationConstants is a "default" lookup section. If App Map references are not found in other named sections then ApplicationConstants will be searched.

In addition, if a variable is referenced in a test table and it has never been assigned a value then the framework searches the ApplicationConstants section of the "current" or "default" Application Map. (Yes, more than one can be open and in use at any given time.)

Consequently, this simple TIDTest.MAP shows that we have provided default CONSTANT values for ^var1 and ^var2 referenced in the *Concatenate* Driver Command record in the TIDTest we just reviewed (and re-viewable here).

Sample TIDTest.MAP:

```
[ApplicationConstants]
Var1="abc"
Var2="def"
```

Execute the runTIDTest Robot Script

It is now time to actually run this simple test. But don't get yourself all excited about it. There won't be anything to see. No fancy application. No GUI exercising. And, in fact, this test is going to fail.

Remember (or listen up), this test was primarily designed to verify the installation of our SAFS framework tools. Since we have not yet enabled RRAFS to use the SAFS framework we can expect the test to report several problems and ultimately that the test has failed.

Don't worry, in very short order we will be enabling SAFS and we will then see this test pass.

So go ahead, from the RRAFS101 project have Robot execute its "**runTIDTest**" script and we will see what shows up in our log.

Review the Robot TIDTest Log

Having executed the "**runTIDTest**" script in the RRAFS101 project we should now have a **TIDTest.Robot.txt** log in **RRAFS101\Datapool\Logs**. Go ahead and open that log in your favorite text editor\viewer or review the online version here.

Note that you will find an "Unimplemented" command warning for the *Concatenate* Driver Command. The Warning is issued because the Concatenate keyword is supported by one of the external SAFS engines and we have not enabled those yet.

The *OnNotEqualGotoBlockID* Driver Command does indeed branch to the "*ERROR*" block because we did NOT Concatenate our two variables. Consequently, the comparison for equality has failed.

Finally, having transferred execution to the ERROR block we log a test failure and reach the end of our test. As mentioned previously, all these "problems" are to be expected for this particular test until we enable SAFS.

You can close the log. We are done with this section. Perhaps you should take a break. If you proceed on the normal course path we will next review the contents and settings available to us in the RRAFS.INI configuration file.

Sample TIDTest.Robot.txt:

```
================================================================================
SQARobot Test Log: C:\RRAFS101\Datapool\Logs\TIDTest.Robot.txt
Version 1.1
Log OPENED 11-22-2004 00:00:10
  ------  START DATATABLE: C:\RRAFS101\Datapool\TIDTest.cdd
          Application Map set to>C:\RRAFS101\Datapool\TIDTest.MAP
- WARNING Unknown Driver Command in table C:\RRAFS101\Datapool\TIDTest.cdd at Line  3
          C, Concatenate,"abc","def", "result"
          OnNotEqualGotoBlockID test values did not Match. Attempting branch to "ERROR"
          TRANSFERRING EXECUTION TO BLOCKID "ERROR" in table
C:\RRAFS101\Datapool\TIDTest.cdd at line 10
          Begin Block "ERROR"
**FAILED**Error in one or more SAFS modules!

  ------  END DATATABLE: CYCLE TABLE: C:\RRAFS101\Datapool\TIDTest.cdd
 ======== BEGIN STATUS REPORT: C:\RRAFS101\Datapool\TIDTest.cdd
             TOTAL RECORDS: 5
           SKIPPED RECORDS: 0

              TEST RECORDS: 1
             TEST FAILURES: 1
             TEST WARNINGS: 0
              TESTS PASSED: 0

          GENERAL FAILURES: 0
          GENERAL WARNINGS: 1
               IO FAILURES: 0
 ======== END STATUS REPORT: C:\RRAFS101\Datapool\TIDTest.cdd

Log CLOSED 11-22-2004 00:00:10
```

Overview of RRAFS.INI

We are finishing up this whole "Enabling Rational Robot" topic with a discussion of the RRAFS.INI file. By default, we don't have to do anything with this file. The traditional simple operation of RRAFS will continue just fine even if RRAFS.INI were purged from the system. But it is thru the RRAFS.INI file that we begin to take advantages of the most exciting features available to RRAFS!


To start with, you should be able to find that there is a RRAFS.INI file in your DDE_RUNTIME directory. (Remember where DDE_RUNTIME is? If not, review the RRAFS101 Definitions.) This copy of the INI file is called the "default RRAFS.INI" file, or simply the "default INI" file.


Every Rational Project we properly enable per our earlier instructions will also have a RRAFS.INI file in the Project Runtime directory. (The Datapool\Runtime directory within each Project, remember?) This copy of the INI file is called the "Project RRAFS.INI" file, or simply the "Project INI" file.

At this time RRAFS will only seek out "RRAFS.INI" files. The user cannot yet specify a configuration file of a different name. A separate Project RRAFS.INI file could be prepared for each desired test configuration. Prior to commencing tests for that configuration the current Project RRAFS.INI file would be replaced with the one containing the desired settings. Batch files that automate this RRAFS.INI swapping provide a good option for this limitation.

You can refer to this RRAFS.INI file for the following discussions.

Sample RRAFS.INI:

```
;Configuration file for SAFS Engine for Rational Robot.
;
;RRAFS.INI can be configured and placed in the DDE_RUNTIME directory
;for settings that affect ALL projects; and/or, it can be configured and
;placed in each project's Datapool\Runtime directory for
;project-specific settings.

;DIRECTORIES can be configured in project's Datapool\Runtime\RRAFS.INI
;file so that only that project is affected.

[DIRECTORIES]
;BENCHDIR=<fullpath or project relative path>
;DATADIR=<** cannot yet be modified at this time **>
;DIFFDIR=<fullpath or project relative path>
;LOGDIR=<fullpath or project relative path>
;TESTDIR=<fullpath or project relative path>

;=====================================================================
;SAFSVARS Tool Options
;AUTOLAUNCH -- Launch if needed and not running (TRUE/FALSE/YES/NO).
;OPTIONS    -- Options or PARMS for the service commandline
;              Normal use requires no parameters.
;=====================================================================
[SAFSVARS]
AUTOLAUNCH=FALSE
;OPTIONS=
```

(Sample RRAFS.INI continues...)

```
;=====================================================================
;SAFSMAPS Tool Options
;AUTOLAUNCH -- Launch if needed and not running (TRUE/FALSE/YES/NO).
;OPTIONS    -- Options or PARMS for the service commandline
;              By default will use current Datapool\ directory.
;=====================================================================
[SAFSMAPS]
AUTOLAUNCH=FALSE
;OPTIONS=

;=====================================================================
;SAFSLOGS Tool Options
;AUTOLAUNCH -- Launch if needed and not running (TRUE/FALSE/YES/NO).
;OPTIONS    -- Options or PARMS for the service commandline
;              By default will use current Datapool\Logs directory.
;=====================================================================
[SAFSLOGS]
AUTOLAUNCH=FALSE
;OPTIONS=

;=====================================================================
;SAFS/DriverCommands options
;AUTOLAUNCH -- Launch if needed and not running (TRUE/FALSE/YES/NO).
;JVM        -- Full path to an alternate java.exe (JVM).
;              otherwise Java is assumed to be in the System PATH.
;CLASSPATH  -- Alternative classpath when launching Java.
;              Otherwise, Java's default classpath usage in effect.
;HOOKCLASS  -- Alternate DriverCommands hook,
;              Default: org.safs.DCJavahook
;OPTIONS    -- Options for the hook's commandline:
;              "LOG" -- option for developer debugging
;=====================================================================
[SAFS_DRIVERCOMMANDS]
AUTOLAUNCH=FALSE
;JVM=<path to java.exe>
;CLASSPATH=<alternate classpath for use with Java -cp argument>
;HOOKCLASS=org.safs.DCJavaHook
;OPTIONS=LOG


;DIFFER can be configured in DDE_RUNTIME directory RRAFS.INI to
;affect all projects.  Consult documentation in NTCommandUtilities
;for more information.

[DIFFER]
;TOOL_EXE=DIFF.EXE
;TOOL_OPTIONS="--text -s -y"
;TOOL_BINARY_OPTIONS="--binary -s"

;DIFFVIEWER can be configured in DDE_RUNTIME sqabas32\RRAFS.INI to
;affect all projects.    Consult documentation in NTCommandUtilities
;for more information.

[DIFFVIEWER]
;TOOL_EXE=GEMINI.EXE
;TOOL_OPTIONS=" "
```

(Sample RRAFS.INI continues...)

```
;=====================================================================
;SAFS/RobotJ options
;AUTOLAUNCH -- Launch if needed and not running (TRUE/FALSE/YES/NO).
;JVM        -- Full path to an alternate java.exe (JVM).
;              otherwise Java is assumed to be in the System PATH.
;CLASSPATH  -- Alternative classpath when launching Java.
;              Otherwise, Java's default classpath usage in effect.
;INSTALLDIR -- Full path to the Rational com.rational.test.ft.wswplugin
;DATASTORE  -- root directory for RobotJ project datastore,
;              Path relative to the Robot project or the full path.
;PROJECT    -- TestManager/Robot project .rsp file,
;              .rsp filename for the project or the full path to the
;              appropriate TestManager .rsp file.
;BUILD      -- TestManager Build for logging
;HOOKSCRIPT -- RobotJ Script that launches the SAFS Hook.
;              The script created by user during RobotJ setup that
;              simply has CallScript("TestScript"); (or equivalent).
;LOGFOLDER  -- TestManager LogFolder ("Default" by default)
;LOG        -- Alt Log filename to use. Will use HOOKSCRIPT by default.
;USERID     -- Userid used to login to TestManager project.
;PASSWORD   -- Password used to login to TestManager project (if any).
;OPTIONS    -- Other Options for the hook's commandline:
;              "LOG" -- option for developer debugging
;=====================================================================
[SAFS_ROBOTJ]
AUTOLAUNCH=FALSE
;JVM=<path to java.exe>
;CLASSPATH=<alternate classpath for use with Java -cp argument>

;COMMENT OUT THE UNNEEDED INSTALLDIR. VERIFY THE PATH IS RIGHT FOR YOUR
INSTALLATION
;---------------------------------------------------------------------------
;INSTALLDIR="C:\Program
Files\Rational\RobotJ\hshell\plugins\com.rational.test.ft.wswplugin"
INSTALLDIR="C:\Program
Files\Rational\XDETester\eclipse\plugins\com.rational.test.ft.wswplugin_2.0.0"

DATASTORE=DatastoreJ
PROJECT=DDEngine.rsp
BUILD="Build 1"
HOOKSCRIPT=TestScript
LOGFOLDER=Default
LOG=TestScript
USERID=canagl
PASSWORD=
;OPTIONS=
```

Default vs. Project RRAFS.INI Files

So why do we need two (or more) RRAFS.INI files, you ask? Well, think about it. There are likely settings that apply to everything we do on this machine with RRAFS (there are). And there may be some settings that are unique to each Rational Project (right again). And lastly, there may be some settings that we typically use for all testing but we occasionally wish to override in one Project or for one test. That is why we can have more than one RRAFS.INI file.

Now there is an order to the madness, here! The default RRAFS.INI file in the DDE_RUNTIME directory should contain just that: default settings that will be applied to ALL Rational Projects used by the machine. This file will be searched last whenever an INI value is sought.

The machine-global settings that are typically found here would be those for the DIFFER and DIFF_VIEWER and possible some of the SAFS Framework settings if the SAFS Framework is going to be used in all projects. These will be discussed in greater detail in the following pages.

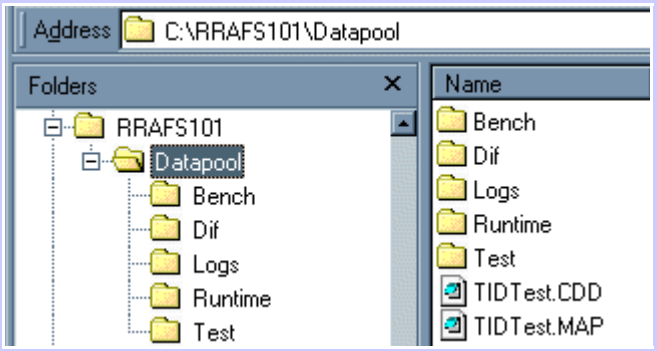
The Project RRAFS.INI file in the Project's Datapool\Runtime directory is used to provide project-specific settings. Values here will override any values stored in the default RRAFS.INI file because this file is searched first. For example, the default INI file typically has all the SAFS tools "AUTOLAUNCH" settings set to "FALSE". This disables the enhanced SAFS Framework *except* for those projects you enable via the Project INI files.

You will actually be enabling the SAFS Framework when we reach the topic "*Enabling SAFS for RRAFS*". So don't fret just yet about any of that.

The optional **DIRECTORIES** settings, though rarely used, are very often unique to a project and we will go over this section of the RRAFS.INI file next.

DIRECTORIES Settings in RRAFS.INI

The RRAFS.INI file **[DIRECTORIES]** settings allow us to specify custom paths that override the default Bench, Test, Dif, and Logs subdirectories used in each RRAFS Project. At this time, we cannot override the parent "Datapool" or "Datapool\Runtime" directories via the configuration file because we require the default Datapool\Runtime project path to find the INI file settings in the first place! (Breathe)



If you look in the Project repository we enabled earlier you can see these Datapool subdirectories that can be changed via the INI file:

- Bench
- Dif
- Logs
- Test

These values are sought only when RRAFS is initializing. That is typically when you run the script that launches the framework.

So why would we ever want to change these? Well, some folks just prefer a customized location for their assets. But the most common use for this is to facilitate configuration testing.

For configuration testing we run the same test against different sytem configurations -- languages, operating systems, servers, databases, or whatever. The tests would share the same static Datapool test tables and possibly the Bench(mark) files, but each test could store separate runtime logs, test data (actuals compared to benchmarks), and diffs (comparator differences) into unique directories.

There are probably 3.5 million other reasons individuals might desire to change these, but most folks never do. That is why these settings remain commented by default. If you wish to change a setting, uncomment the appropriate one and provide the path to the directory you wish to use. This directory MUST already exist and the user logged onto the machine MUST have write access to the directory.

DIFFER and DIFF_VIEWER Settings

In the RRAFS.INI file there are sections for **[DIFFER]** and **[DIFF_VIEWER]**.

By default RRAFS currently uses the Windows **FC.EXE** file comparator to perform both text and binary differences. However, this built-in utility is not aging well with the newer Windows operating systems and faster computers. (*I\O induced false failures are becoming quite common for some people on certain machines.*)

For this reason, RRAFS allows the user to install an alternative tool like ExamDiff (and\or others) and configure RRAFS to use it via the **DIFFER** settings.

In RRAFS, file comparison operations that fail generally create a convenience batch file for post-test execution. This batch file is written to the active Dif folder and will invoke the predefined DIFF VIEWER -- the program that can display two text files simultaneously and visually show the differences. The BAT file will be of no use if no viewer is installed.

By default RRAFS uses ExamDiff as the DIFF VIEWER. However, *ExamDiff is not provided by SAFSDEV*. For this reason, RRAFS allows the user to install an alternative viewer and configure RRAFS to use it via the DIFF_VIEWER settings.

The following links to relevant RRAFS documentation might be useful to review if you find yourself using either of these features:

- RRAFS Diff Tool Execution
  (http://safsdev.sourceforge.net/sqabasic2000/NTCommandUtilities.htm#run_difftool_exe)
- RRAFS Text Comparisons
  (http://safsdev.sourceforge.net/sqabasic2000/NTCommandUtilities.htm#run_rrafstextdiff_exe)
- RRAFS Binary Comparisons
  (http://safsdev.sourceforge.net/sqabasic2000/NTCommandUtilities.htm#run_rrafsbinarydiff_exe)
- RRAFS NTCommandUtilities
  (http://safsdev.sourceforge.net/sqabasic2000/NTCommandUtilities.htm)
- GNU DiffUtils and Doc
  (http://unxutils.sourceforge.net) (http://www.fsf.org/software/diffutils/manual/diff.html)

SAFS Framework Settings Overview

Finally, in the RRAFS.INI file there are a number of settings we will collectively call the "SAFS Framework" settings. Generally, they enable or disable the use of the external SAFS Framework tools which includes Rational XDE Tester. These are disabled by default.

The SAFS Framework sections of the RRAFS.INI file:

- **SAFSVARS**
- **SAFSMAPS**
- **SAFSLOGS**
- **SAFS_DRIVERCOMMANDS**
- **SAFS_ROBOTJ**

We will be going over these in significant detail in the later topic, "*Enabling SAFS for RRAFS*". So for now, go grab yourself some Cheeze snacks and a drink and slap yourself conscious.

Quiz: Enabling Rational Robot

After Robot is installed, and after we have installed RRAFS, there are still some Robot settings and optimizations to make. Let's see how much you remember from the lesson!

1 1 Marks

Robot settings modified for RRAFS are found in which 2 Robot menus?

Answer:

- ☐ a. Tools->General Options
- ☐ b. Tools->GUI Record Options
- ☐ c. Tool->Extension Manager
- ☐ d. Tools->GUI Playback Options

2 1 Marks

The activities in RRAFS allow us to reduce the playback delays between command and keystrokes. What is the recommended setting for both Delay Between Commands and Delay Between Keystrokes?

Answer:

- ☐ a. 10 milliseconds
- ☐ b. 25 milliseconds
- ☐ c. 50 milliseconds
- ☐ d. 0 milliseconds

3 1 Marks
RRAFS provides a new set of logs often used instead of TestManager's log. However, RRAFS still needs the Log Management setting 'Output Playback results to log' enabled (checked), True or False?
Answer: ○ True ○ False

4 1 Marks
RRAFS recommended settings suggest that "Detect Unexpected Active Window" should remain enabled (True or False)?
Answer: ○ True ○ False

5 1 Marks
In order for RRAFS Error Recovery to work as designed what setting is recommended for "On Failure to Remove Unexpected Active Window"?
Answer:
○ a. Skip Current Script
○ b. Continue Execution
○ c. Abort Playback

6 1 Marks
RRAFS provides for a more robust Error Recovery mechanism than the default mechanisms provided by Robot. What is the recommended Error Recovery setting for Script Command Failures and Verification Point Failures?
Answer:
○ a. Abort Playback
○ b. Continue Execution
○ c. Skip the Current Script

7 1 Marks
In order to ensure the best overall performance of RRAFS you should disable all unneeded Tool Extensions in the Extension Manager (True or False)?
Answer: ○ True ○ False

8 1 Marks
What is the quickest method to enable an existing Robot Project repository with the subdirectories needed by RRAFS?
Answer:
○ a. Create the Datapool directory and subdirectories manually
○ b. Run a script that creates the directory structure
○ c. Copy the SAFS\Project subdirectories structure

9 1 Marks
The SAFS\Project template has a Robot script that can be used to run a RRAFS install sanity check. What is the name of this Robot script?

Answer:
- (○) a. runTIDTest.rec
- ( ) b. runTIDTest.bat

10 1 Marks
In order to see App Map edits between Robot runs it is recommended the DDGClearAppMapCache function appear at the top of each script launching a RRAFS test (True or False)?

Answer: (○) True ( ) False

11 1 Marks
Where is the 'Default' RRAFS.INI configuration file located?

Answer:
- ( ) a. The RRAFS project's Datapool directory
- (○) b. DDE_RUNTIME: .\Rational\Rational Test\sqabas32 directory
- ( ) c. The Robot project's root directory
- ( ) d. The RRAFS project's Datapool\Runtime directory

12 1 Marks
Where is the overriding Project RRAFS.INI optionally located?

Answer:
- ( ) a. The RRAFS project's Datapool directory
- ( ) b. DDE_RUNTIME: .\Rational\Rational Test\sqabas32 directory
- ( ) c. The Robot project's root directory
- (○) d. The RRAFS project's Datapool\Runtime directory

This Page Intentionally Blank

Lesson

4

RRAFS Framework Features

Go ahead, try to say that fast 3 times!
Now that we've got it, what does it do? What is it that RRAFS has bestowed upon our wonderous Robot that wasn't there before?

My! How We've Grown!
RRAFS Framework Features
Quiz: RRAFS Framework Features

My How We've Grown

Growing into Test Portability and Tool Multiplicity

I think one thing that set RRAFS apart was that it was the culmination of input from many different sources. Public design and implementation discussions were a large part of the preliminary work for RRAFS. Rational Robot users in many different parts of the world actively sought to contribute code for this project. Even users of competing tools recognized there was something exciting afoot, and some sought to build compatible frameworks for their own toolset.

RRAFS provides a unique open source solution for keyword-driven test automation that otherwise just does not exist. Large chunks of new functionality have been provided by users to satisfy needs that were previously unmet. In the earliest days, calls for help were often answered with, "RRAFS can't do that just yet." But those types of replies are now few and far between.

And RRAFS ain't just for IBM Rational Robot anymore!

RRAFS can now call other "engines" and take advantage of external commands and tools like SAFS/DriverCommands. It can work side-by-side with XDE Tester to use the best of both tools. More tools will soon be available as SAFS engines and RRAFS will be able to use those, too.

The portability and multiplicity of RRAFS and SAFS execution does not limit us to choosing one tool over another. While we can choose to migrate from one tool to another with relative ease, the framework is most feature-rich and robust when we use multiple tools at the same time in the execution of the same tests. And that is something no other GUI test automation framework currently offers.

Carl Nagle
Project Manager, SAFSDEV

Exploring the RRAFS Framework

The RRAFS Framework was made by Automators for Automators! Besides providing a keyword-driven testing framework, RRAFS provides other features and functions you cannot readily do with Robot by itself. RRAFS does not just provide keywords that wrap standard Robot functions. For example, we can check for the existence and state of Windows menus BEFORE we attempt to act on them. For some hard-to-match items we provide for partial substring matches when seeking things like HTML links and Tree nodes!

Exploring the SQABasic Libraries

See all that is exposed to traditional scripting in SQABasic Libraries. When you launch that online reference you can see that it is largely just a collection of links to the documentation for the many varied libraries provided by RRAFS. This one document should be considered a good reference point for RRAFS automators. This is a good one to Bookmark.

SQABasic Libraries provides links to nearly every critical document there is for RRAFS. Including, at the bottom, Component Function Script Library documentation not generally linked from anywhere else.

This component function documentation is NOT the same documentation provided in the keyword reference docs -- although much of the doc overlaps. This documentation is actually what should be considered the API doc for the functions themselves, not the keywords that invoke them. And these component function docs will contain API info for functions that are available to scripts and are NOT available via keywords (though the functions are used in the implementation of keywords).

Some of the most interesting general-purpose libraries are listed below:

ProcessContainer	MenuUtilities	StringUtilities	WIN32Utilities
LogUtilities	DDVariableStore	DDGUIUtilities	WINRegistryUtilities
ApplicationUtilities	DDUtilities	Publish	XMLUtilities

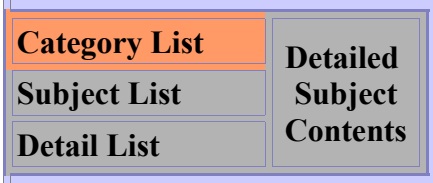
SQABasic Libraries
http://safsdev.sourceforge.net/sqabasic2000/SQABasicLibraries.htm

Component Function Script Library
http://safsdev.sourceforge.net/sqabasic2000/SQABasicLibraries.htm#ComponentFunctions

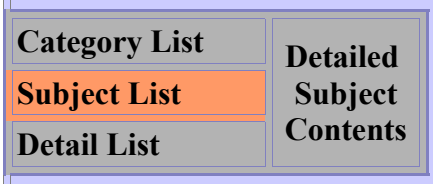
Exploring the SAFS Keyword Reference

The *SQABasic Libraries* of the previous section primarily linked us to the functions and routines implemented throughout the many Rational Robot libraries. When we are in the midst of writing keyword-driven tests, however, we need something that tells us about all the available keywords and their usage. One of the sources for that is the all-inclusive SAFS Keyword Reference.

Compare the real Keyword Reference in your web browser (launched from the link above) as we discuss the various panels and features shown below. Experiment with using the panels as we discuss each one in turn.



Category List : This is where we begin by selecting the broad category of what we want to review. When we select a category the *Subject List* is populated with the list of subjects for the selected category. Categories 'Component Functions′ and 'Driver Commands' contain our keyword reference.



Subject List : The list of subjects for the selected Category. For example, Component Functions subjects are listed according to component type. Driver Command subjects are also grouped according to similar functionality. When the SAFS Reference is first loaded it defaults to the 'Component Functions' list. When we select a subject from the Subject List the Detail List is populated according to the information provided by the subject. This may be a list of available keywords or some other pertinent information.

Detail List: The list of detailed content available for the selected Subject. For example, the list of keywords available for the selected component type. When the SAFS Reference is first loaded the Detail List for the GenericMasterFunctions Component Functions is listed. This is because these keywords or commands are generally available for ALL components. When we select an item in the Detail List we load the detailed information into the Detailed Subject Contents panel.

Detailed Subject Contents : This is where the detailed information we have selected via the other panels is finally viewed.

SAFS Keyword Reference
http://safsdev.sourceforge.net/sqabasic2000/RRAFSReference.htm

Explore 'Other References'

Category List	Detailed Subject Contents
Subject List	
Detail List	

Notice in the Category List two other entries besides the Component Functions and Driver Commands keyword categories. These are:

- Other References
- How to Interpret This Doc

When you select **'Other References'** you will see the Subject List panel update with a list of broad categories of subjects. If you select various items within this updated Subject List you will find the Detail List panel update with links to various docs. Go ahead and select each item in the Subject List and see what shows up in the Detail List for each item.

The keen observer will notice that nearly every document available from the SQABasic Libraries document we saw earlier can be found in these links. Also, many of the documents available on the SAFSDEV Home Page can be found in these links. Thus, the SAFS Keyword Reference can become a one-stop reference for most of the documentation available for RRAFS and SAFS.

SQABasic Libraries
http://safsdev.sourceforge.net/sqabasic2000/SQABasicLibraries.htm

SAFSDEV Home Page
http://safsdev.sourceforge.net

Using the Keyword Reference

Category List	Detailed Subject Contents
Subject List	
Detail List	

When we select 'Component Functions' or 'Driver Commands' from the SAFS Reference Category List we are referencing the documentation for the predefined keywords provided by all possible implementations -- not just RRAFS. That is an important piece of information to keep in mind.

This one document attempts to expose ALL implemented keywords. That means keywords known to Rational Robot, Mercury WinRunner, Rational XDE Tester, and others. When you think you have found a keyword you want to use, make sure the engine you are using has implemented support for the keyword.

Select **'Component Functions'** in the *Category List* to get a Subject List of all component function subjects.

Select **'JavaTreeFunctions'** in the *Subject List* to get a Detail List of all keywords predefined for Java Trees.

Notice the right side *Detail Subject Contents* panel has not changed during any of this. That panel will not update until you have made a selection that explicitly updates the panel. Note the Detail List's **'JavaTreeFunctions'** title is a link that will update the contents panel as will each of the listed keywords.

On the next page will discuss the colors and icons.

Interpreting the Keyword Reference

Category List **Subject List** **Detail List**	**Detailed Subject Contents**

Notice in the Detail List panel we should have the list of available JavaTreeFunctions keywords. Some are red, but most are blue. You will also find to the right of each keyword a set of icons. You may have to drag the vertical splitter bar to the right to expose the icons or scroll the Detail List to the right.

As should be written above the list of keywords, any keyword colored red indicates it is a 'deprecated' keyword. In other words, it is considered 'obsolete' and has likely been replaced by another keyword. Deprecated keywords will still work, but you should avoid using them whenever possible. Normally the detailed information provided for any deprecated keyword will list which replacement keyword is preferred.

Select the '**Collapse**' keyword in the Detail List and when the main contents panel updates you should see that its preferred replacement is 'CollapseTextNode'.

The icons to the right of each keyword indicate which SAFS engines have implemented support for the keyword and what type of support has been implemented. The best way to get the skinny on the icons is to view the Legend.

Select '**How To Interpret This Doc**' in the Category List and you will find a full explanation of each icon and how the different icon colors indicate what type of support exists in each engine.

Using Multiple SAFS Reference Windows

Category List	Detailed Subject Contents
Subject List	
Detail List	

The fact that this SAFS Reference uses HTML Frames affords us some interesting possibilities. For one thing, we are not limited to these 4 predefined panels displayed in one browser window. And we are not limited to just one Subject List or Detail List. If we open our lists into new windows then we can keep any number of these lists open and in view at any one time!

The following instructions assume Internet Explorer. Similar functionality should be available in other browsers but may require different user actions to activate.

Right-Click the '**Component Functions**' link in the Category List and select 'Open in New Window' from the popup menu.

Now do the same thing with the '**Driver Commands**' link so that BOTH of these Subject Lists are available in separate windows.

We can now very quickly and easily lookup different Component Functions and Driver Commands without having to repeatedly wait for the Subject List to update. And this works for the Detail Lists too. For example, we might want to keep some frequently visited component function lists open in separate windows for ready reference.

Of course, it can get 'messy' with many open windows. But you are free to use as many or as few as you want. And while you're thinking about that, think about stretching or grabbing yourself some wakeup juice before you move on to the next section.

Exploring the DDEngine Reference

The "older" version of the keywords reference. Unlike the highly modular SAFS Keyword Reference, all keyword data is downloaded in one rather large gulp for the DDEngine Reference. (A high-bandwidth connection is pretty much required for this beast!)

All is not as it seems with the DDEngine Reference. This highly scripted doc actually presents much of the data in a rather unremarkable and inconspicuous hierarchical format. Hidden layers are exposed by clicking underlined text. (These will look like but not act like normal HTML links.)

The data for this doc comes from the same XML files used to create the SAFS Keyword Reference, but it is presented in a different format in a single document download.

If you have the bandwidth for the one-time download, go ahead and explore this doc and see how it works. It may prove to be a valuable reference aid at times when repeated calls to the web server are not always successful (but that should be rare).

Note that a local version of the DDEngine Reference doc is created in the SAFS\doc directory when XSLBuildDDEngineReference.BAT is executed from the SAFS\bin directory of the SAFS Framework install.

SAFS Keyword Reference
http://safsdev.sourceforge.net/sqabasic2000/RRAFSReference.htm

DDEngine Reference
http://safsdev.sourceforge.net/sqabasic2000/DDEngineReference.htm

Exploring Application Maps and Application Constants

Application Maps are the essential piece that makes test maintenance easier and most productive. They map the user-friendly component names and references we use in our test tables to the cryptic information needed by our automation tools so that they may correctly locate and manipulate GUI components.

```
MainWindow="Type=Window;Caption={Notepad*}"
```

The Test Automator will spend a great deal of their time creating, editing, and perfecting robust Application Maps.

RRAFS Application Maps not only provide the mentioned mapping mechanism, but they also enable the user to store named constant values called Application Constants that can be referenced like any valid DDVariable. Additionally, some commands have optional or required parameters -- such as "where" to click -- that can be stored in the Application Map as a named reference like "Top", "Bottom", "Center", or "TitleBar". The documentation for individual commands will state whether or not this mechanism is available and the particular syntax needed for the reference.

```
Center="50,50"
```

There is also a very flexible look-thru mechanism employed by the framework to locate items in the Application Map. Component definitions not found within their parent Window definition will then be sought as global Application Constants. Thus, for example, "OK" buttons that exist in many windows with an identical definition or recognition string can actually be defined just one time and stored as a single Application Constant used by all windows.

```
OKButton="Type=Pushbutton;Name=cmdOK"
```

So let's now explore some of these features in greater detail...

Application Map Format

The format of the Application Map is a flat text file in a Windows INI file format. That means that the file is broken up into sections uniquely identified within brackets as shown below:

```
[MainWindow]
MainWindow="Type=Window;Caption={Notepad*}"
Editor="Type=EditBox;Index=1"

[AboutWindow]
AboutWindow="Type=Window;Caption={About*}"
CloseButton="Type=Pushbutton;Text=Close"
```

As you can see, each section generally defines a Window within the application and within each window we define the names and recognition methods of each component. Note that the window itself requires an entry just like the child components.

(Though the format is called the "Windows INI format", the multi-platform SAFSMAPS service in the SAFS Framework can read this format on any supported platform.)

Application Map Component Definitions

```
[MainWindow]
MainWindow="Type=Window;Caption={Notepad*}"
Editor="Type=EditBox;Index=1"

[AboutWindow]
AboutWindow="Type=Window;Caption={About*}"
CloseButton="Type=Pushbutton;Text=Close"
```

As shown above the component definitions are nothing more than Name=Value pairs. The example shows each Value enclosed in quotes. That is not strictly required in most cases, but highly recommended.

Again notice that within each window section the window component itself must have an entry providing its own recognition string.

Application Map Component Parameter References

```
[MainWindow]
MainWindow="Type=Window;Caption={MyApp*}"
ToolBar="Class=AfxCustomToolBar;Index=1"

[ToolBar]
CopyButton="Coords=7,10"
PasteButton="Coords=25,10"
```

Some command keywords allow or require parameters stored in the Application Map as a named reference. The GenericObject CLICK command is one example. The App Map above shows a ToolBar in the MainWindow whose custom nature perplexes Robot Classic. However, the Generic CLICK command allows us to provide meaningful named references for specific items or locations on the ToolBar. This allows for meaningful records like those shown below:

```
T  MainWindow  ToolBar  Click  CopyButton
T  MainWindow  ToolBar  Click  PasteButton
```

Consult the documentation for each keyword that supports this feature, but in general the named reference is normally stored in a section with the same name as the component being addressed -- even if it isn't a window -- as is shown in our sample App Map above.

GenericObject CLICK Command

http://safsdev.sourceforge.net/sqabasic2000/GenericObjectFunctionsReference.htm#detail_Click

Application Map [ApplicationConstants]

```
[ApplicationConstants]
UserID="LongJohnSilver"
Password="PegLeg"
OKButton="Type=PushButton;Text=OK"

[LoginWindow]
LoginWindow="Type=Window;Caption=Login"
UserField="Type=EditBox;Name=cmdUser"
PasswordField="Type=EditBox;Name=cmdPassword"

[AboutWindow]
AboutWindow="Type=Window;Caption=About"
```

The predefined ApplicationConstants section of an App Map provides 2 benefits:

1. Global Predefined "Constants"
2. Shared Component Definitions

Some example records taking advantage of these benefits:

```
T  LoginWindow  UserField  SetTextValue  ^UserID
T  LoginWindow  PasswordField  SetTextValue ^Password
T  LoginWindow  OKButton  Click
...
T  AboutWindow  OKButton  Click
```

Note how the constant values ^UserID and ^Password are referenced just like any normal DDVariable. We do not "set" the value of these DDVariables because they will be found as constants in the App Map as long as they reside in the [ApplicationConstants] section.

Also note that we have referenced the OKButton in both the LoginWindow and the AboutWindow even though neither of those window definition sections contain an OKButton definition. The lookup mechanism knows to go search the [ApplicationConstants] section if a sought reference cannot be found in its expected section.

Application Map Dynamic Component Recognition

Finally, there are often times some components in an application that cannot readily be predefined in an Application Map. For example, there may be a dynamic number of Checkbox options, RadioButtons, or Images on a Web Page. And sometimes components are endowed with a Name or an ID that changes every single time the application is run. After you have gone and shot the developer, you still must deal with the problem.

It is usually very difficult, tedious, or even impossible to provide predefined App Map definitions for these objects. For these situations we must resort to dynamic component recognition.

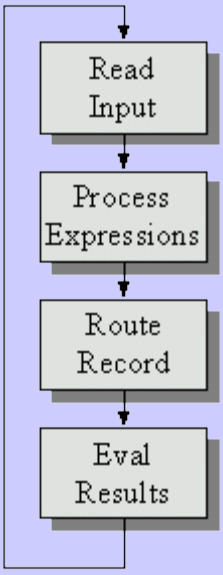
```
[WebPage]
WebPage="Type=Window;Caption=Image Maps"
Image="_DDV:"
AltImage="_DDV:AnotherImage"
```

When our App Map lookup encounters the predefined "_DDV:" text it knows the value sought is actually stored -- usually dynamically calculated at runtime -- in a DDVariable. If no DDVariable name is specified immediately after the "_DDV:" prefix then the name of the DDVariable is the same as the name of the component. Any text following the "_DDV:" prefix is considered to be the name of the DDVariable holding the sought value.

```
C  SetVariableValues  ^WhichOne="1"
C  SetVariableValues  ^ImagePrefix="Type=HTMLImage;Index="

B  Loop
C  SetVariableValues  ^Image=^ImagePrefix & ^WhichOne
C  SetVariableValues  ^AnotherImage=^ImagePrefix & (^WhichOne + 1)

T  WebPage  Image     Click
T  WebPage  AltImage  Click

C  SetVariableValues  ^WhichOne=^WhichOne + 2
C  OnNotEqualGotoBlockID  Loop  ^WhichOne   "11"
```

Above we have a loop that will click on the first 10 Images in WebPage even though we only have 2 defined in the App Map. We dynamically create the "Image" and "AltImage" recognition strings in DDVariables and the App Map lookup will grab these when the information is requested. Note how the App Map definition for component "AltImage" specified the use of DDVariable ^AnotherImage instead of the default ^AltImage.

Exploring DDVariables and Expression Handling



The graphic at left shows a simplified flow of test processing performed by RRAFS and all SAFS Drivers.

Read Input: retrieve the next test record from the currently running test table.

Process Expressions: evaluate and resolve all DDVariables and expressions in the test record so that the test record is reduced to nothing but literal text.

Route Record: send the test record to appropriate processor functions based on the contents of the record. For example, Driver Command records (**C**) are routed to different processors than Tests and Component Function records (**T**).

Eval Results: increment various pass\fail counters and check for other conditions before we finally go back and start the process all over again with the next test record.

So why the heck did we just go through THAT?!

Well there is an important concept to be realized here: a SAFS engine processor -- like those for Driver Commands and Component Functions -- will never see a DDVariable or an expression in a test record. They will only see literal text. The DDVariables and expressions are resolved first and foremost before ANYTHING else of significance can happen! You will see this in action later when we actually trace through the driver code that does this.

OK? OK. Now what in tarnation is a DDVariable you ask? Let's talk about that next.

Data-Driven Variables -- DDVariables

Back in 1998 when RRAFS was known as the "DDE" or "Data-Driven Engine" we introduced "Data-Driven Variables" or "DDVariables". These are not like your typical program variables in Robot. They are not even specific to RRAFS.

DDVariables are stored globally and are available to all running processes on the machine. Thus, RRAFS code in SQABasic can access these variables and so can any other program or application on the machine. This is all done behind the scenes through ActiveX\COM objects from the DDVariableStore.DLL. We don't have to worry about any of this, though, because it is all handled automatically within RRAFS.

On the other hand, later when we enable the SAFS Framework these DDVariables are no longer confined to processes and programs on the local machine. They become accessible to remote machines as well through the SAFSVARS service!

An important thing to deduce from this is that all DDVariables have global scope. That means all our test tables are dealing with the same shared DDVariables. After all, that is the whole purpose behind DDVariables -- to transfer and share data across boundaries. Those with programming experience should know this global scope is both a blessing *and* a curse.

If we are not careful with defining and using these DDVariables we might have some of our scripts or test tables changing the values unexpectedly. This can lead to unexpected behavior that is difficult to isolate and correct.

Important DDVariable details:

1. Global Scope: Shared by all processes on the machine
2. Transfer data between test tables
3. Transfer data between test tables and scripts
4. Transfer data between scripts
5. Transfer data with other applications
6. Ultimately, transfer data with other machines

Now let's explore how we go about getting and setting DDVariables.

DDVariables in Test Tables

In SAFS test tables we reference DDVariables by prefixing a caret (^) to a variable name. Variable names are generally alphanumeric with many other characters that can legally be part of the name. However, variable names cannot contain embedded whitespace, quotation marks, assignment symbols (=), and most other operators used in expressions. Periods and underscores are acceptable as part of a variable name. The first occurence of whitespace or an assignment symbol will mark the end of the variable name. Variable names are NOT case-sensitive.

Examples of legal variable references:

^AVariable	^User.ID	^Client1_Name
^anotherVariable	^User.Password	^Client1_Address

Any field in a valid test record can contain a DDVariable -- even fields beyond those required by whatever command is present in the test record. A valid test record is any record not skipped or treated as an ignored comment.

We set DDVariables by assigning them values -- even values stored in other DDVariables:

^Var1 = "Some Value"	^Var2 = 1	^var3 = ^var1

Using the above example ^Var3 will contain the value "Some Value" when it is assigned the value stored in ^Var1. ^Var2 contains the string "1" since all values -- even numbers -- are stored as strings.

If the caret or variable name is enclosed in quotes then it is NOT treated as a variable at all. It is simply treated as literal text and left unchanged:

"^IsNotAVariable"	"^IsNotAnAssignment=Anything"

The above examples are nothing more than literal strings because they are enclosed in double-quotes.

DDVariables Processing

When EXPRESSIONS (discussed soon) are not enabled we are dealing with simple DDVariable processing. The DDVariable reference or assignment will only be recognized and processed if the caret is the first non-whitespace character in the test record field.

^Var1	DDVariable Reference
^Var1 = "Some Value"	DDVariable Assignment
"^Var1 = nothing"	Literal Text Only
A bogus ^var1 reference	Literal Text Only

The latter two examples above are treated as literal text because they are either enclosed in quotes or the caret normally prefixing a DDVariable is NOT the first character in the field and thus is considered part of a literal text string.

With simple DDVariable processing that is all we can do with a DDVariable. We can reference a DDVariable value in a field or we can assign a value to a DDVariable in a field. We cannot do anything else in that field. This insures the caret and other characters can be used as literal text readily without worrying about character escaping.

To get fancier, the tester can enable EXPRESSIONS and do a whole lot more. But before we learn more about Expressions, let's review how DDVariables might be used in traditional scripts.

DDVariables in Scripts

As we mentioned previously, scripts can get and set these global DDVariables too. This allows data to be transferred between test tables and scripts, between scripts, and -- when the SAFS Framework is enabled -- between scripts and any number of other external tools.

The important SQABasic library for handling DDVariables in scripts is DDVariableStore.SBL. Review that document and find the routines for:

- DDVGetVariableValue
- DDVSetVariableValue

There is also a document specifically for Using DDVariables in scripts!

Perhaps now we are ready to learn a little more about Expressions...

DDVariableStore.SBL
http://safsdev.sourceforge.net/sqabasic2000/DDVariableStore.htm

Using DDVariables in Scripts
http://safsdev.sourceforge.net/sqabasic2000/UsingDDVariables.htm#in_scripts

Enabling Expressions

If you remember a few frames back we mentioned that "simple" variable processing only allows you to get and set variable values. This is the default mode of operation for the RRAFS framework.

The framework does support a limited number of expression operators when Expressions are enabled, but Expressions are NOT enabled by default. We enable and disable Expressions in our test tables with a Driver Command:

```
C  Expressions  ON
C  Expressions  OFF
```

Scripts can also enable and disable Expressions via the SQABasic Global Variable DDU_EXPRESSIONS_MODE in DDUtilities:

```
DDU_EXPRESSIONS_MODE = 1   'ON
DDU_EXPRESSIONS_MODE = 0   'OFF
```

DDU Expressions Mode in DDUtilities
http://safsdev.sourceforge.net/sqabasic2000/DDUtilities.htm#globals

Expressions Operators

In RRAFS, the processing of Expression strings occurs in StringUtilities.SBL. So scripts can also process expressions using the same routines used by the RRAFS drivers.

The following operators are supported:

^	(Caret/Circumflex) Variable Prefix
=	Assignment operator
"	A single Double-Quote mark
&	(Ampersand) String concatenate operator
+	Addition operator
-	Subtraction operator
*	(Asterisk) Multiplication operator
/	Division operator
%	(Percent) Modulus/Remainder operator
(	Open Group operator
)	Close Group operator

Of particular note are the parenthesis -- the group operators. Variables, strings, and expressions in groups are separately processed before anything else in the overall (complete) expression. In fact, groupings are initially processed as separate expressions and can even contain separate assignment operations.

So let's see review some sample expressions...

StringUtilities.SBL
http://safsdev.sourceforge.net/sqabasic2000/StringUtilities.htm#processexpression

Expression Groups and String Concatenation

( ) -- Groupings are denoted by enclosing portions of the overall expression inside matching parenthesis. Groupings can be embedded inside other groupings and the embedded grouping is processed before the outer groups. As mentioned before, groupings are processed recursively as standalone expressions and can thus contain assignment operations.

^Var1 = (^Var2 = 1 & (^Var3 = 14)) & 1

The grouping containing ^Var3 will be processed first and ^Var3 will be assigned the value of "14". The grouping containing ^Var2 is processed next as:

(^Var2 = 1 & "14")

Thus, ^Var2 is assigned the value of "114". Once the groupings have been processed we are left with:

^Var1 = "114" & 1

Finally, ^Var1 is assigned the value of "1141".

So, as you can see, the concatenation operator treats everything as a string -- even numbers. This is convenient because even numbers are stored as strings in variable storage.

So lets see the same expression values processed using numeric operators...

Expression Numeric Operations

We saw that the concatenation operator will concatenate even numeric values strictly as literal strings. Well, the numeric expression using numeric operators will logically produce a different result:

^Var1 = (^Var2 = 1 + (^Var3 = 14)) + 1

The innermost grouping containing ^Var3 is still processed first and still resolves to "14" since this is strictly an assignment and all variables are stored as string values. The next grouping containing ^Var2 is then seen as:

(^Var2 = 1 + "14")

Since the (+) operator is numeric the value stored in ^Var2 is "15". This leaves all groupings processed and an expression like this:

^Var1 = "15" + 1

Thus, ^Var1 is assigned the value of "16".

Lastly, it is important to note that numeric operators expect their operands to be numbers. Missing operands, empty strings, variables with no assigned value, and non-numeric literal text will be converted to the numeric value "0".

Note the progression of processing on this type of an expression:

^Var1 = (^Var2 = "Carl") + 1
^Var1 = "Carl" + 1
^Var1 = "1"

Variable ^Var2 will be assigned the value "Carl", but when the resulting expression for ^Var1 is evaluated the (+) operator will convert the operands to numeric values and the literal text "Carl" will be converted to "0" since it has no valid numeric value. Thus, ^Var1 = "0" + 1 and results with ^Var1 receiving the value of "1".
(Note, the numeric converion of "Carl" to "0" does not change the stored value of ^Var2.)

Expressions and Quotation Marks

Expressions and operators are only processed when they are not embedded inside or recognized as literal strings. We can specify literal strings by enclosing them in a pair of double-quote marks (**"**).

^Var1 = "(0 + 1) + 2"

^Var1 will be assigned the literal unmodified string "(0 + 1) + 2" -- without the quotes -- since the operators are embedded inside the quoted string and are thus ignored.

Finally, you can actually include a double-quote mark inside a literal string by using two double-quote marks back-to-back inside the quoted string:

^Var1 = "He said ""Hi!"""

^Var1 will be assigned the value: He said "Hi!"
Each paired set of double-quotes embedded inside a quoted string indicates that a single double-quote is to be included in that string at that position.

Beware Malformed Expressions

Malformed Expressions are generally those that are improperly formatted or missing expected operators. For example, a grouping with an opening parenthesis but no closing parenthesis, or a literal text string with an opening quote mark but no closing quote mark, or an incorrect number of quote marks when trying to embed quote marks in literal text. These are evil!

In general, RRAFS and the SAFS Framework will attempt to process as much as it can but will give up at some point when presented with a malformed expression. This means the expression may be returned only partially processed or not processed at all. Variables may not have been assigned the values we expected or they may not be assigned values at all.

The results of malformed expressions are not guaranteed and are subject to change with future releases.

Exploring Standard and Custom Counters

Each RRAFS driver -- CycleDriver, SuiteDriver, and StepDriver -- has global status counters available. These are global variables of user-defined type AUStatusInfo. The AUStatusInfo UDT is defined in Application Utilities. The globals associated with the drivers are not normally used directly by scripts. These are normally used only by the core libraries and internal functions:

CycleDriver	CycleDriverInfo, CycleDriverTestInfo
SuiteDriver	SuiteDriverInfo , SuiteDriverTestInfo
StepDriver	StepDriverInfo
Script Access	**ScriptGUIInfo, ScriptStatusInfo**

Typically, scripts called from RRAFS will reference the ScriptGUIInfo and ScriptStatusInfo structures only. RRAFS will have copied the appropriate driver globals into the Script Access globals. This allows the script to reference the appropriate Script Access globals without concern over whether it was invoked by CycleDriver, SuiteDriver, or StepDriver. When the script returns, RRAFS will copy the information back into the appropriate driver globals.

There are also user-defined counters at your disposal in Application Utilities. These can be created and manipulated by scripts or by RRAFS test tables via DriverCounterCommands. With these commands you can start and stop individual counters, suspend and resume all counters, copy or store counter values in DDVariables, and even output a special Status Report into the log for individual counters.

Application Utilities UDT
http://safsdev.sourceforge.net/sqabasic2000/ApplicationUtilities.htm#user_defined

ScriptGUIInfo, ScriptStatusInfo
http://safsdev.sourceforge.net/sqabasic2000/ApplicationUtilities.htm#globals

Using Custom Counters

Example in Test Tables using DriverCounterCommands:

```
C   StartCounter        "AppSpec1.1"   "Login with Valid Credentials"
...
C   StopCounter         "AppSpec1.1"
C   LogCounterInfo      "AppSpec1.1"   "Login with Valid Credentials"
C   StoreCounterInfo    "AppSpec1.1"   ^varPrefix="AppSpecVar"
...
C   SetVariableValues   ^failures=^AppSpecVar.test_failures
```

Example in Script using ApplicationUtilities and LogUtilities:

```
status = AUStartStatusCounter ( "AppSpec1.1" )
...
status = AUStopStatusCounter ( "AppSpec1.1" )
status = AUGetStatusCounterStatus ( "AppSpec1.1", ScriptStatusInfo )
LULogStatusInfo ScriptStatusInfo, MainLog, "Login with Valid Credentials Status"
status = AUVariableStoreStatusInfo ( "AppSpec1.1", "AppSpecVar" )
```

DriverCounterCommands
http://safsdev.sourceforge.net/sqabasic2000/DDDriverCounterCommandsIndex.htm

Application Utilities
http://safsdev.sourceforge.net/sqabasic2000/ApplicationUtilities.htm#austartstatuscounter

LogUtilities
http://safsdev.sourceforge.net/sqabasic2000/LogUtilities.htm#lulogstatusinfo

Exploring Error Recovery and Flow Control

Believe it or not, sometimes things don't always go as planned. Sometimes the test environment isn't quite right. Sometimes the automation gets out-of-sync. Sometimes the application we are testing is actually...uhhh..."bad"!

For these scenarios, and an untold number not mentioned here, RRAFS provides the means to automatically respond to various conditions. Basically, we tell RRAFS the names of specific BlockIDs we want executed when specific test conditions exist and RRAFS will automatically branch to those blocks when the test condition is encountered.

The DriverFlowCommands used for this recovery mechanism are listed below:

SetExitTableBlock	Block for forced ExitTable cleanup/recovery
SetGeneralScriptFailureBlock	Block for Failure cleanup/recovery
SetInvalidFileIOBlock	Block for Invalid IO cleanup/recovery
SetNoScriptFailureBlock	Block for "OMG! It worked!" handling
SetScriptNotExecutedBlock	Block for "Command Not Executed" handling
SetScriptWarningBlock	Block for Warning cleanup/recovery

When conditions invoke a recovery block we can then execute whatever cleanup or recovery we want. We can call other tables, suites, or even scripts. Note that execution does not ever return to the point in the table that invoked the recovery mechanism. An explicit branching command like GotoBlockID will be needed if you wish to retry the test steps that originally invoked the recovery mechansim.

On the next page we will provide an example table using this mechanism.

DriverFlowCommands
http://safsdev.sourceforge.net/sqabasic2000/DDDriverFlowCommandsIndex.htm

Error Recovery and Flow Control Example

Below we have an example that will attempt to execute a block of code. If the execution is not successful it will retry up to 3 times before finally giving up. This example is not the most efficient means to accomplish this, but it serves the purpose of demonstrating the use of the error recovery mechanism. Normally, you would enable error recovery across a larger block of code in which the error might occur on any line at unexpected times.

```
C,  SetVariableValues,  ^try = "0",  ^abort = "3"
C,  SetGeneralScriptFailureBlock,  "RetryBlock"

B,  "Loop"
T,  CheckApplicationState

B,  "CleanupBlock"
C,  SetGeneralScriptFailureBlock, ""
C,  ExitTable

B,  "RetryBlock"
C,  CallScript,  "ResetApplicationState"
C,  OnEqualGotoBlockID,  "CleanupBlock", ^try = ^try + 1, ^abort
C,  GotoBlockID, "Loop"
```

The RetryBlock is only invoked IF the CheckApplicationState suite returns a failure. If no failure occurs we simply disable/reset the error recovery mechanism and exit the table. We are happy.

If the RetryBlock is invoked by failure we will execute a ResetApplicationState script. We then increment our ^try counter and see if it is equal to our ^abort count. If these are equal we will branch/abort to the CleanupBlock and exit the table. If these counts are not yet equal we will branch up to the Loop block and CheckApplicationState again.

Working with and Calling Scripts

RRAFS provides some excellent support for integrating traditional scripts into the keyword-driven architecture. This is not limited to the availability of Global DDVariables. This also includes access to current test info, status info, and the ability of scripts to add to this information and provide important test feedback.

We are first going to explore the "CallScript" Driver Command that allows our keyword-driven test to invoke a tradional script. We will then review some important Global SQABasic User-Defined Type structures made available to the script for this invocation. Finally, we will go over what is called the "Implied CallScript" mechanism. This allows RRAFS to invoke a script even without using the CallScript Driver Command.

Let's go ahead and get started with the explicit execution of the CallScript Driver Command...

"CallScript" -- Explicit Script Execution

Below are several examples of invoking a traditional script with the "CallScript" Driver Command. Commas are used to delineate fields in the record:

1. C, CallScript, "MyLoginScript"
2. C, CallScript, "MyLoginScript", ^userid="John", ^password="Smith"
3. C, CallScript, "MyLoginScript", "John", "Smith"

Example #1 simply invokes a Robot script named "MyLoginScript". There are no other parameters or values passed in the record that invokes the script. (Though, DDVariables and Application Constants in the App Map are available to the script always.)

Example #2 invokes the "MyLoginScript" and provides 2 DDVariable parameter values within the record that invokes the script. With the exception that the ^userid and ^password DDVariables exist with the specified values, there really is no difference between Example #2 and Example #3 by the time the script is called. The DDVariables have already been assigned values and those values now appear in the record as literal text identical to Example #3.

Example #3 shows that we can simply pass literal text values to the script without resorting to DDVariables. The entire input record will be available to the script for processing. Note that foregoing the use of DDVariables means the literal text is likely tied to a specific field locations and cannot be arbitrarily moved to different fields.

In Example #3, the userid "John" is bound to field#4 in the record and cannot be moved to field#5 or #6 without refactoring the script accordingly. The use of DDVariables in Example #2 allows the userid and password to be specified in any arbitrary order. They can even be specified in any preceding record or script and need not appear in this record at all -- as might be the case in Example #1.

So just how does the script gain access to the input record and other vital RRAFS information? How does the script provide status information back to RRAFS upon completion? Very smart questions! Let's go find out.

ScriptGuiInfo and ScriptStatusInfo Global Storage

The global ScriptGUIInfo and ScriptStatusInfo variables provide a temporary bridge between RRAFS and your test scripts. These are global SQABasic UDT variables of type AUGUIInfo and AUStatusInfo as defined in ApplicationUtilities.

The information in ScriptGUIInfo and ScriptStatusInfo is very transient and is only valid for the one call to a script. A script should copy these structures to local variables if the script will be reinvoking the engine in any way. Be sure to copy the locally stored copies back to ScriptGUIInfo and ScriptStatusInfo before exiting the script.

Examine the data available in the AUGUIInfo global ScriptGUIInfo from the link above. The elements of this structure most often used by scripts are shown below:

- ScriptGUIInfo.inputrecord
- ScriptGUIInfo.separator
- ScriptGUIInfo.fac
- ScriptGUIInfo.statuscode

.inputrecord -- provides the script with the entire record used to invoke it. This will have all fields including those the script may be designed to use as input and/or output parameters in place of, or in addition to, DDVariables.

.separator -- the separator needed to delimit the .inputrecord fields. The .separator is used in various StringUtility calls to extract individual field/parameter values from the .inputrecord.

.fac -- the LogFacility used for the currently running test. The script will use this .fac when logging messages via LogUtilities. These messages will appear in all enabled logs just like all messages logged by RRAFS.

.statuscode -- the script should set the .statuscode appropriately to a valid DRIVER RETURN CODE value as defined in DDUtilities Constants. If the script wishes to record multiple test counts instead of just one, the script will need to increment the ScriptStatusInfo (or the local copy) accordingly via the appropriate AUIncrement functions in ApplicationUtilities.

A sample script using these features follows on the next page.

ScriptGUIInfo, ScriptStatusInfo
http://safsdev.sourceforge.net/sqabasic2000/ApplicationUtilities.htm#globals

AUGUIInfo, AUStatusInfo
http://safsdev.sourceforge.net/sqabasic2000/ApplicationUtilities.htm#user_defined

DRIVER RETURN CODES
http://safsdev.sourceforge.net/sqabasic2000/DDUtilities.htm#constants

AUIncrement Functions
http://safsdev.sourceforge.net/sqabasic2000/ApplicationUtilities.htm#declarations

Sample Script using ScriptGUIInfo and ScriptStatusInfo

Lets use the sample script invocations discussed earlier:

- C, CallScript, "MyLoginScript", ^userID="John", ^pass="Smith"

Here is a sample "MyLoginScript" implementation using the ScriptGUIInfo and ScriptStatusInfo and other functions typically used by scripts:

```
Sub Main
  With ScriptGUIInfo
    status = DDVGetVariableValue( "userid", theUserID)
    status = DDVGetVariableValue( "pass"   , thePassword)

    'alternative .inputrecord field mechanism
    theUserID     = GetTrimmedQuotedField( .inputrecord, 4, .separator)
    thePassword = GetTrimmedQuotedField( .inputrecord, 5, .separator)

    status = PerformLogin( theUserID, thePassword)

    if status = sqaSuccess then
       AUIncrementTestPasses ScriptStatusInfo
       LogMessage "Login Successful!", .fac, PASSED_MESSAGE
       .statuscode = DDU_NO_SCRIPT_FAILURE
    else
       AUIncrementTestFailures ScriptStatusInfo
       LogMessage "Login Failed!", .fac, FAILED_MESSAGE
       .statuscode = DDU_GENERAL_SCRIPT_FAILURE
    end if

  End With
End Sub
```

Scripts via an Implied CallScript mechanism

Any unrecognized record type will trigger an attempt to execute an Implied CallScript command. That is, we will check to see if the unknown record type is actually the name of a script to execute. The purpose behind this is to allow an efficient use of fields when calling scripts that pass a larger number of parameters in the record.

1. C, CallScript, RunLoginScript, ^user = "John", ^pass = "Smith"
2. RunLoginScript, ^user = "John", ^pass = "Smith"

Notice that record #2 is identical to record #1 with the exception of the first 2 fields. Both of these records will attempt to locate and execute the script "RunLoginScript". Record #2 will do this because it does not recognize "RunLoginScript" as a valid record type (C, T, S, etc..). Since it does not recognize the record type it assumes it must be the name of a script.

An important thing to keep in mind when using these two mechanisms to invoke scripts is the difference in field locations for any passed parameters. Since these two records both pass DDVariables this is not an issue. However, if the script attempts to locate the value for parameters USER and PASS with GetTrimmedQuotedField then the field locations are different between these two records:

```
1. theUserID     = GetTrimmedQuotedField ( .inputrecord, 4, .separator )
   thePassword = GetTrimmedQuotedField ( .inputrecord, 5, .separator )

2. theUserID     = GetTrimmedQuotedField ( .inputrecord, 2, .separator )
   thePassword = GetTrimmedQuotedField ( .inputrecord, 3, .separator )
```

For this reason, it is often better for maintenance to pass parameters to scripts via DDVariables instead of relying on hardcoded field locations in the .inputrecord.

Outside the Box with the SAFS Framework

As you can see in this lesson, the RRAFS Framework offers a wealth of features and functions not available anywhere else -- far and above what IBM Rational Robot can do out-of-the-box. And many testers are quite satisfied with that.

But there is so much more!

The next topic in this RRAFS101 Tutorial will go into much greater detail on what the larger SAFS Framework offers RRAFS. There is also the SAFS101 Tutorial which you may or may not have reviewed already. If you haven't seen it you must remember to go through that tutorial. It's a "moral imperative!"

The SAFS Framework allows us to break out of the confines of using *only* IBM Rational Robot and RRAFS. The SAFS Framework provides a mechanism to access IBM Rational XDE Tester via the SAFS/RobotJ engine using the same keyword-driven test tables and to even run XDE Tester scripts from Rational Robot. We can also access new Driver Commands implemented in Java via the free opensource SAFS/DriverCommands engine and we can implement links to other testing tools and programming environments.

We can make new commands or scripts in Java, VB, C/C++, .NET, Perl, and other scripting languages and call them in our test tables or Robot scripts. We don't have to limit ourselves to just one testing tool and one testing language. Did we say all this already?! Do we sound excited about this?! Think about that for a while -- don't just read the words.

So hurry up! Finish this RRAFS Framework Features lesson and move on to Enabling SAFS for RRAFS!
AAAAAAHHHHHH !!!!

The SAFS101 Tutorial
http://safsworks.com/online/course/view.php?id=3

Quiz: RRAFS Framework Features

Were you paying attention throughout the long lesson on framework features? Let's see how much of this stuff sunk in!

1 1 Marks
One HTML document contains links to virtually all available to Robot users. This is NOT the Keyword themselves. What is the name of this document?

Answer:
- a. ApplicationUtilities.htm
- b. WIN32Utilities.htm
- c. SQABasicLibraries.htm
- d. DDUtilities.htm

2 1 Marks
The SAFS Keyword Reference also has links to most (perhaps all) of the same documentation via its "Other References" link (True or False)?

Answer: True / False

3 1 Marks
Which of the following docs is the primary keywords reference doc using FRAMES to link many low-bandwidth docs on demand?

Answer:
- a. Data-Driven Engine Reference
- b. SAFS Quick Reference
- c. The SAFS Reference aka the SAFS Keywords Reference

4 1 Marks
Which reference doc pulls down ALL the keyword reference information into one interactive TreeView-like document with one big high-bandwidth download?

Answer:
- a. The SAFS Keywords Reference
- b. The Data-Driven Engine Reference aka the DDEngineReference
- c. The SAFS Quick Reference

5 1 Marks
In the default SAFS\bin directory there is a DOS batch file that can build local versions of the DDEngineReference and the SAFS Reference docs in the SAFS\doc directory. What is the name of this batch file?

Answer:
- a. XSLBuildDDEngineReference.bat
- b. XSLQuickReference.bat
- c. XSLComponentActionsMap.bat

6 1 Marks
Select the 3 primary functions of an Application Map:

Answer:
- [ ] a. Map user-friendly component names to automation tool references
- [ ] b. Create Busy work for Test Automators
- [ ] c. Store named Application Constants as DDVariable References
- [ ] d. Store named parameter references required by some keywords.
- [ ] e. Store dynamic DDVariable values

7 1 Marks
What is the file format for Application Maps?

Answer:
- ○ a. TAB delimited text files
- ○ b. Windows INI File Format
- ○ c. Java Properties-like Name=Value pairs.

8 1 Marks
To support dynamic component recognition, we can specify in the App Map that a components recognition string is stored in a DDVariable. What special prefix is used on the Value side of the App Map component's Name=Value pair to indicate the actual value is stored in a DDVariable?

Answer:

9 1 Marks
Keyword implementations must resolve DDVariable values before they can perform their action (True or False)?

Answer: ○ True ○ False

10 1 Marks
DDVariables are machine-global -- shared by all scripts, test tables, and tools on the local machine (True or False)?

Answer: ○ True ○ False

11 1 Marks
When using STAF and the SAFS Framework our machine-global DDVariables are visible to other STAF-enabled machines (True or False)?

Answer: ○ True ○ FALSE

12 1 Marks
In a test record, what character prefix identifies an item as a DDVariable?

Answer:
- ○ a. The Percent Sign: %
- ○ b. The Caret or Circumflex: ^
- ○ c. The Dollar Sign: $
- ○ d. The Ampersand: &

13 1 Marks
By default, we can only get and set DDVariables values -- we do not process complex expressions. What Driver Command keyword accepting an "ON" or "OFF" parameter can be used to enable expressions that use mathematical and string concatenation operators?

Answer:

14 1 Marks
Which SQABasic Library implements the functions for handling User-Defined Counters?

Answer:
- ○ a. ApplicationUtilities
- ○ b. LogUtilities
- ○ c. DDUtilities
- ○ d. StepDriver

15 1 Marks
Which set of Driver Commands exposes user-defined counters via keywords?

Answer:
- ○ a. DDDriverLogCommands
- ○ b. DDDriverCommands
- ○ c. DDDriverCounterCommands
- ○ d. DDDriverStringCommands

16 1 Marks
Which set of Driver Commands provide the bulk of the command keywords for Error Recovery and Flow Control?

Answer:
- ○ a. DDDriverFlowCommands
- ○ b. DDDriverCommands
- ○ c. DDDriverStringCommands
- ○ d. DDDriverDeprecatedCommands

17 1 Marks
Select the 2 global User-Defined Type (UDT) variables Robot scripts can use to exchange runtime data and status information with the RRAFS Drivers:

Answer:
- ☐ a. AUGUIInfo
- ☐ b. ScriptStatusInfo
- ☐ c. ScriptGUIINfo
- ☐ d. AUStatusInfo

18 1 Marks
RunLoginScript, ^userid="John", ^pwd="Smith"

The Driver will not recognize a command like this because there is no recognizable Record Type in field #1. The Driver will attempt to see if there is a "RunLoginScript" that can be executed. What type of record or command is this?

Answer:
- ○ a. Custom Test Record
- ○ b. Custom Record Type
- ○ c. Implied CallScript Command

Lesson

5 Using RRAFS Process Container

One of the fundamental tasks for defining SAFS tests is the development of the Application Map. The RRAFS ProcessContainer is the tool of choice for this or any task where we need to examine components and component properties with Rational Robot. Let's explore the varied ways in which we can use this excellent tool!

Using RRAFS Process Container

Quiz: Using RRAFS Process Container

Process Container: The Quicker Fixer-Upper

Using RRAFS Process Container

What exactly is Process Container?

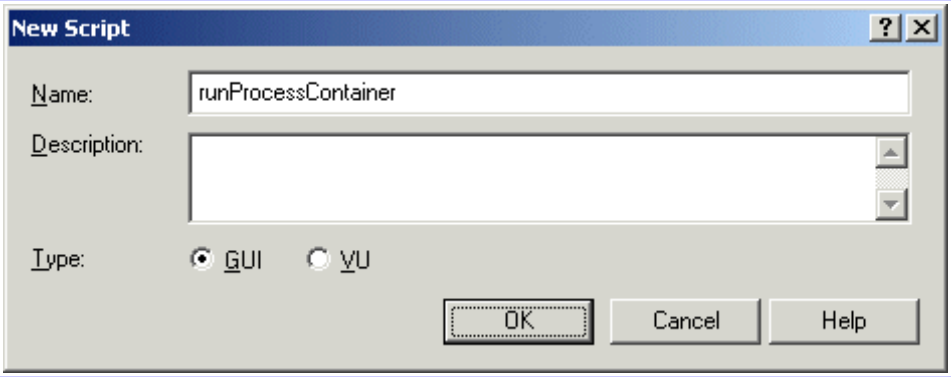
What is Process Container?
We hear this question often. Is it a 3rd party tool separate from Robot? Is it part of Robot? Is it Inspector? Is it a script or library? Well, let's go ahead and get these all out of the way.

RRAFS Process Container is simply a complex SQABasic program wrapped with a SQABasic Dialog Box -- something we don't see everyday in Rational Robot scripts. It is, essentially, a Robot function that can recursively call SQAGetChildren and SQAGetProperties on a target window and output the information to a file. As such a function, it can only be called from a Robot script.

Of course, Process Container allows us to tweak what is done and how it is done through the SQABasic dialog box that is presented.

In order for us to use Process Container our Robot project must have a script that will launch it. This script must be created or imported into each Robot project that will run Process Container.

Create a New Script in Robot called "runProcessContainer". The entire script is comprised of the 4 lines shown below:



The script:

```
'Include "DDEngine.SBH"
Sub Main
  ProcessContainer
End Sub
```

Notice the function we execute is called 'ProcessContainer'. You cannot name the script 'ProcessContainer' or the script will be recursively calling itself until the system runs out of memory. Save the new script, compile it, and we'll move on...

ProcessContainer: http://safsdev.sourceforge.net/sqabasic2000/ProcessContainer.htm

Select the Type of Window Client for Process Container to Process

Go ahead and launch the runProcessContainer script in Robot. You should soon see the Robot: Process Container Dialog. You can launch and view a local copy of the Process Container online doc by clicking the Help button at the bottom of the dialog.

Before we proceed on some of the details, it is important to recognize why we have the ComboBox at top-right for selecting the different client types:

- Window Client
- Web Client
- Java Client

Take, for example, your typical web browser. Robot sees this Window in two completely different ways. If you ask to view the Browser Window hierarchy there is a standard Win32 Window with Toolbars, Statusbar, and other Window-type components. When you look in this hierarchy there is no web application -- just Window objects.

On the other hand, we normally care more about the actual web application or HTML contents. So we have to tell Robot through different recognition strings that we want the web content (Web Client), not the Browser components (Window Client).

Among other internal processing effects, setting the appropriate client type provides the novice user with a default recognition string that should work for the selected client type with just a simple tweak identifying the target Window's Caption.

Let's review a recommended methodology for using Process Container.

Use Process Container Methodically for Best Results

Usually, the most productive way to use Process Container is to follow these steps. Each step is generally a separate run of Process Container with different settings, or different recognition strings:

1. **Process a Window for Object Hierarchy**
   - Provide a unique 'Window\Object Name' for output
   - Enable 'Process Children' if not too complex
   - Disable 'Process Properties'
   - 'Remove Parent Info' almost always selected for Web and Java
   - Optionally 'Process Menu', but just once per Window
     (Note: non-standard Menus will NOT be seen by 'Process Menu'.)
   - Press OK and wait for Output Files
2. **Process an individual object for Object Properties**
   - Use recognition strings found in previous run
   - Update 'Object Recognition Method' to specify desired object
   - Provide a unique 'Window\Object Name' for output
   - Disable 'Process Children'
   - Enable 'Process Properties'
   - Press OK and wait for Output Files.
   - Repeat as desired for other 'interesting' child objects
3. **Optionally Create/Append an App Map file**
   - Do this only once per Window or multiple Appends occur.
   - Disable 'Process Children', if desired.
   - Disable 'Process Properties'
   - Enable 'Remove Parent Info' (usually) for Web and Java
   - Enable 'Append AppMap'
   - Specify a new or existing AppMap file to Append
   - Press OK and wait for Output Files.

Consult the Process Container online doc for details on using each dialog control.

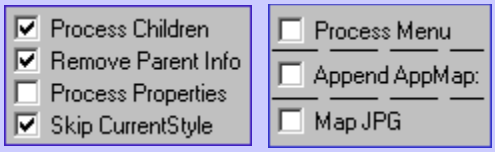
Use Process Container to evaluate Object Hierarchy

Refer to the Robot: Process Container Dialog. You can also view the Process Container online doc by clicking the Help button.

(*Note: This exercise assumes you have your HTML extension enabled in Robot's Tools->Extension Manager settings. If you don't then you will need to enable this and close and restart Robot.*)

Run the ProcessContainer script if Process Container is not already running. Make sure the dialog settings are as shown below. These settings are for capturing this very page! So if the displayed Caption is different on your browser titlebar make the appropriate changes in your entry.

At top select:	**Web Client**
Window Recognition Method:	**Type=Window;Caption={RRAFS101*}**
Object Recognition Method:	**Type=HTMLDocument;Index=1**
Window/Object Name:	**RRAFS101**



Click the **OK** button and Process Container will start collecting all the web client hierarchy information that it can. The Process Container dialog box will disappear and will not reappear until processing is finished. If you have Robot in the background you may be able to see the Console output generated by Process Container.

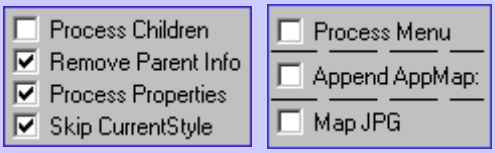
Once Process Container has finished it will actually attempt to launch a Notepad window with the **RRAFS101Obj.txt** data collected. This file will remain in the Project's Datapool directory until deleted or overwritten.
(*You can ignore error code 1015 in the output. This is a known Robot bug.*)

Use Process Container to evaluate Object Properties

Now we are going to checkout the Properties of the top HTMLDocument. Process Container can process properties on everything, but that may take a million years. So it is often best to process properties on one component and only when you need the information. Don't waste your time capturing data you don't really need at this time.

Make sure the dialog settings are as shown below. The previous run of Process Container captured all the child objects and their recognition strings. We can use those recognition strings in the *Object Recognition Method* field to investigate each of those child objects. But right now we are going to stick with the top HTMLDocument object.

At top select:	Web Client
Window Recognition Method:	Type=Window;Caption={RRAFS101*}
Object Recognition Method:	Type=HTMLDocument;Index=1
Window/Object Name:	**RRAFS101Doc**



Click the **OK** button and Process Container will start collecting the HTMLDocument properties information. The Process Container dialog box will disappear and will not reappear until processing is finished. If you have Robot in the background you may be able to see the Console output generated by Process Container.

Once Process Container has finished it will actually attempt to launch a Notepad window with the **RRAFS101DocObj.txt** data collected. This file will remain in the Project's Datapool directory until deleted or overwritten.
(*You can ignore error code 1015 in the output. This is a known Robot bug.*)

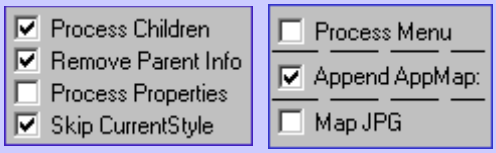
Use Process Container to Build App Maps

Now we are going to Create/Append an App Map with the child information. If we specify an App Map that already exists then the information will be appended to that App Map. If we specify an App Map that does not exist then the App Map will be created.

Remember, you only *Append AppMap* once for the same Window/Screen. Otherwise you will have multiple entries for the same components in the App Map and you will have a larger editing task to remove the duplicates.

Make sure the dialog settings are as shown below.

At top select:	Web Client
Window Recognition Method:	Type=Window;Caption={RRAFS101*}
Object Recognition Method:	Type=HTMLDocument;Index=1
Window/Object Name:	**RRAFS101App**



App Map Filename: **< Keep default or change as desired >**

The default location for App Map files should be the project's Datapool directory.

Click the **OK** button and Process Container will start collecting the HTMLDocument information (again) while it also appends information to the specified App Map.

Once Process Container has finished it will launch a Notepad window with the normal data collected and another with the App Map data. The App Map file will remain in the specified location until deleted.

Process Container will try to give components interesting names, but these are to be edited by the Automator to be more user-friendly. These are the names that will be used throughout all the test tables. So the names should not be changed once set and used in those test tables.

The recognition strings may also have to be tweaked if the Automator finds Robot having difficulty finding the objects with the strings initially captured. Sorry, but App Map development is where the Automator really earns their keep!

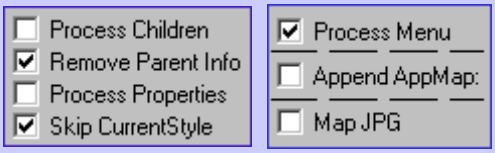
Use Process Container to capture a Window's Menu

Now we are going to attempt to capture Window Menu information. Process Container can only capture standard Window's menus. So new 3rd party types of menus, Java menus, and other oddities will not be "seen" by Process Container. The Menu output file will be blank in this case.

We only *Process Menu* once for the same Window/Screen using the same Menu output filename. If the menu changes during application use then make sure you capture those changes (if desired) in separate runs with different menu output filenames.

Make sure the dialog settings are as shown below.

At top select:	**Windows Client**
Window Recognition Method:	**Type=Window;Caption={RRAFS101*}**
Object Recognition Method:	**\;Type=Window;Caption={RRAFS101*}**
Window/Object Name:	**RRAFS101OL**



Click the **OK** button and Process Container will start collecting data including the Window Menu if it can find one. If your browser is not using a standard menu -- and you will know because the Menu output will be blank -- then you may try running this against a Notepad window or the ClassicsC Online sample application provided by Rational. Of course, you will have to change the Window recognition information in Process Container appropriately to run against those other windows.

Once Process Container has finished it will actually attempt to launch a Notepad window with the normal data and another with **RRAFS101OLMenu.txt** data collected. This file will remain in the Project's Datapool directory until deleted or overwritten.

Quiz: Using RRAFS Process Container

Do you know how to use Process Container? Do you know *why* you should want to use it? Let's check your brainpan and see what you remember...

1 Many people have asked, and we have answered: What is the RRAFS Process Container?

Answer:
- ○ a. An SQABasic Program
- ○ b. A Built-in Robot Inspector
- ○ c. A VisualBasic Application

2 What is the most recommended name for the Robot GUI script used to run the RRAFS Process Container?

Answer:
- ○ a. runProcessContainer
- ○ b. callProcessContainer
- ○ c. CreateAppMap
- ○ d. ProcessContainer

3 When we create or import a new GUI script into a Robot project for running Process Container we should use the script name 'ProcessContainer'.

Answer: ○ True ○ False

4 When creating or appending App Maps, Process Container can write this information to Microsoft Excel.

Answer: ○ True ○ False

5 Select the 3 core client types we can choose in Process Container:

Answer:
- ○ a. VB Client
- ○ b. Windows Client
- ○ c. Web Client
- ○ d. Java Client
- ○ e. HTML Client

6 What is the first or primary step in the recommended usage pattern for Process Container?

Answer:
- ○ a. Map objects to a JPG Screenshot
- ○ b. Process Objects for Object Properties
- ○ c. Process Window for Object Hierarchy
- ○ d. Create/Append App Map files

7 What is the 2nd recommended process step for using Process Container?

Answer:
- ○ a. Process Objects for Object Properties
- ○ b. Create/Append App Map files
- ○ c. Map objects to a JPG Screenshot
- ○ d. Process Window for Object Hierarchy

8 What is the 3rd rec

Answer:
- ○ a. Process Window for Object Hierarchy
- ○ b. Process Objects for Object Properties
- ○ c. Create/Append App Map files
- ○ d. Map objects to a JPG Screenshot

9 Select the 3 items that should be Selected/Checked in Process Container when processing a Web Client for object hierarchy:

Answer:
- ☐ a. Process Properties
- ☐ b. Process Children
- ☐ c. Remove Parent Info
- ☐ d. Skip CurrentStyle

10 Select the 3 items that should be Selected/Checked in Process Container when processing a Web Client for object properties:

Answer:
- ☐ a. Process Properties
- ☐ b. Process Children
- ☐ c. Skip CurrentStyle
- ☐ d. Remove Parent Info

11 Select the 3 items recommended to be Selected/Checked when using Process Container to Append an App Map for a Web Client:

Answer:
- [ ] a. Process Children
- [ ] b. Remove Parent Info
- [ ] c. Process Properties
- [ ] d. Skip CurrentStyle

12 Select the 2 items recommended to be UnSelected/UnChecked when using Process Container to capture a Window Menu:

Answer:
- [ ] a. Process Children
- [ ] b. Process Properties
- [ ] c. Remove Parent Info
- [ ] d. Skip CurrentStyle

13 In which Robot Project directory does Process Container normally write its output files?

Answer:
- ○ a. Datapool
- ○ b. Datapool\Bench
- ○ c. Datapool\Logs
- ○ d. Project Root

Process Container: The Quicker Fixer-Upper

"Robot says it cannot find the object..."
Did you run Process Container to get good recognition strings? How do these strings compare with a recorded script?

"Robot says it cannot find an object property..."
Did you run Process Container to see a list of the valid properties for the object? Is your property in the list? Did you spell it correctly remembering that property names are case-sensitive?

"Robot says it cannot select a Window's MenuItem..."
Did you run Process Container and successfully "Process Menu"? Does your failed menuitem text appropriately match the Process Menu output? How does this text differ from a script recording of selecting the menuitem?
Reminder, Process Menu can only see standard Window Menus.

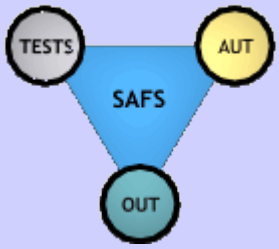
"Robot's Inspector is showing me different info than Process Container..."
Inspector is a different tool from Robot. Inspector sometimes shows values and items that are different from what Robot will see. Use Process Container to see exactly what Robot will see and avoid these Inspector pitfalls.

"My head hurts. I need a break!"
Run Process Container on a complex window with lots of objects making sure that both 'Process Children' and 'Process Properties' are BOTH selected. You may now have a good excuse to take a nice long break! 

This Page Intentionally Blank

Lesson

6



Enabling SAFS for RRAFS

You find yourself happy and proud that RRAFS is working for you. But you've heard here and there of this tool-independent SAFS Framework lurking about that is all the bomb. Now you ask yourself: Where is it? How do I start using it? How is SAFS different than RRAFS? Why would I want to use it?

The lazy answer is: go look at the SAFS101 tutorial! But lazy is not what you're looking for. Besides, SAFS101 doesn't give you the details of how to enable the SAFS Framework with RRAFS specifically. That's where this lesson steps in.

(SAFS must be installed for this lesson. SAFS was installed with a full RRAFS install; or separately via the SAFS101 tutorial mentioned above.)

What does SAFS add to RRAFS?

Enable SAFS for RRAFS

Quiz: RRAFS and the SAFS Framework

SAFS101 Tutorial

http://safsworks.com/online/course/view.php?id=3

What does SAFS add to RRAFS?

Well, let's start with that whole 'lisp' thing. One thing SAFS provides RRAFS users is a built-in and inescapable speech impediment! But we'll look past that right now!

Without SAFS, RRAFS is a self-contained execution engine limited to the capabilities we program into Rational Robot libraries. Yes, Robot can launch other programs and those other programs can do exciting things. But those other programs cannot very readily provide feedback and have Robot react to that feedback.

SAFS provides for access to new testing tools, new Driver Commands, and new Component Functions. SAFS allows Rational Robot to reach out beyond Robot libraries and find new functionality.

For example, RRAFS could always run Robot scripts with the CallScript command. With SAFS we can run scripts from any SAFS-enabled testing tool. Rational XDE Tester is just such a tool. So RRAFS users can also execute Rational XDE Tester scripts using CallScript when SAFS is available!

With SAFS we can implement new Driver Commands and Component Functions in Java or Perl or just about any tool, and these are then available to all testing tools (Robot, XDETester, WinRunner, etc..) RRAFS can use those new Driver Commands and Component Functions and we don't have to write any new Rational Robot code. Rational XDE Tester is an example of this. Component Functions provided by XDE Tester are available to Rational Robot users as well!

You don't have to pick one tool over the other. Use both. Better yet -- use them all!

Now, just try to tell me that isn't cool!

Enabling SAFS for RRAFS

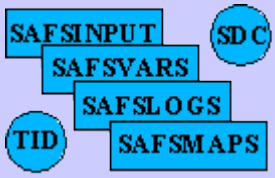
Many Words for Little Work

Do not feel faint of heart because there is so much to see. There is much to see, but little to do. RRAFS tries to handle just about everything automagically for you.

Yet, you need to know how to get started, and how to recover if things don't go as planned.

Prepare To Meet Your Framework

- **Dissect** the **SAFS** directories for tidbits of what is where and why.
- Review the important information in the **Prelude**. It explains how SQABasic scripts will be enhanced to take advantage of SAFS. The Prelude also discusses what happens and what might need to be done if tests are aborted mid-way. You will finish by running a sample test sans SAFS to use for future comparative analysis with a later SAFS-enabled run.
- **Enable SAFS** then provides detailed instructions that actually enable or 'turn on' the SAFS Framework for RRAFS. This also includes running the same sample test and comparing the logs between the run without SAFS, and the one with SAFS enabled.



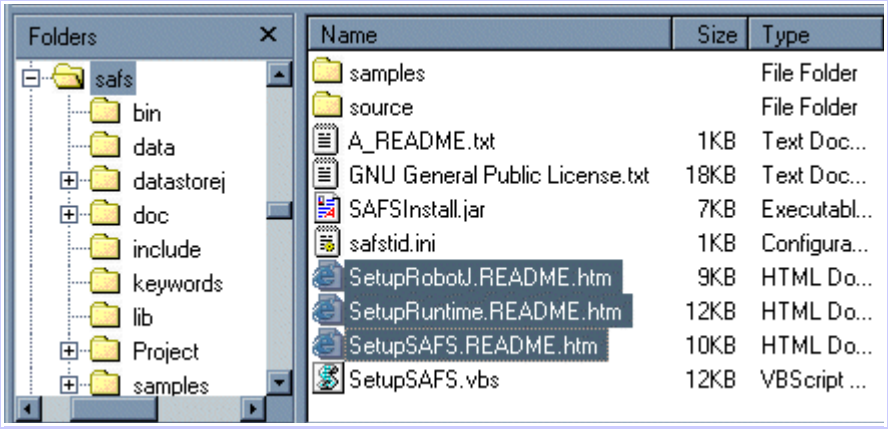
Anatomy of a SAFS Installation

Finally, we are going to see some SAFS stuff! You might want to have a damp towel handy because SAFS is pretty much like super glue. We are going to start pokin' at it and your local version might still be all wet!

Open a Windows Explorer window and navigate to the directory in which SAFS is installed. You should end up with an Explorer window pretty much like this.

SAFS:
Select the SAFS root directory in Windows Explorer. Note there are 3 setup reference files:

1. **SetupSAFS.README.htm**
2. **SetupRuntime.README.htm**
3. **SetupRobotJ.README.htm**



These are informative, and you can review them if you wish, but this entire RRAFS101 tutorial is a detailed exercise in performing the steps documented in those references (and several others). After a quick question, we'll jump ahead to the KEYWORDS subdirectory.

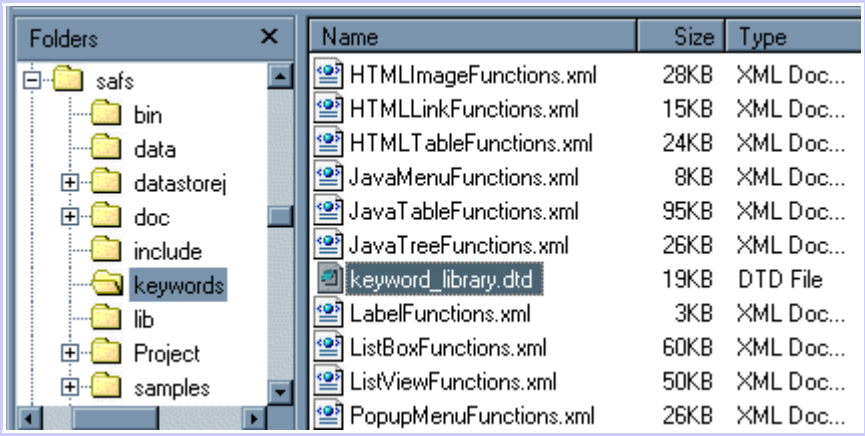
What is the filename of the one INI file in the root SAFS directory?

SAFS\KEYWORDS Subdirectory

SAFS\KEYWORDS:
This is where the excitement is!

All of the Driver Commands and Component Functions along with their parameters and examples are captured in the XML files of this KEYWORDS directory. These are used for an ever-growing number and style of documentation, databases, and data mapping.

You will also find a copy of **keyword_library.dtd** here. This is used to validate the XML for various operations. The DTD makes for an exciting read if you ever become interested in XML and XSL processing. Really!



In an alpha sort of this directory you will find many **XSL*.XSL** and XSL*.XML files at the bottom of the list. These are the XSL Stylesheets and additional XML used in all the various transformations provided for documentation and data. These are used by the BATCH files you will learn about next in the SAFS\BIN directory.

If you get curious enough, viewing the XSL can serve as copy-n-paste samples and templates. The whole of this directory is a pretty cool do-it-yourself primer on using XML and converting XML to text, HTML, and other formats using XSL Stylesheets!

After this quick question, let's move on to the SAFS\BIN directory. You can execute some of these transformations from there!

When sorted by filename, what is the first XML file in this directory?

Open CheckBoxFunctions.XML

Open CheckBoxFunctions.XML in any XML Viewer (like Internet Explorer) or your favorite text editor.

What is the first KEYWORD documented in CheckBoxFunctions.XML?

- ○ UnCheck
- ○ Click
- ○ Check

Still in CheckBoxFunctions.XML...

Last review of CheckBoxFunctions.XML for now...

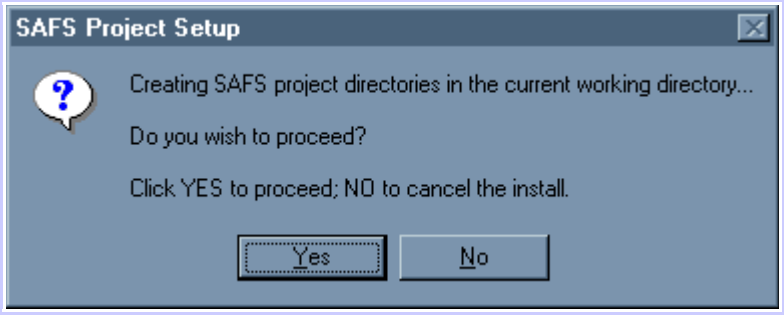
What 3 engines are listed as supporting the CHECK keyword in CheckBoxFunctions.XML?

- ○ WinRunner
- ○ Robot
- ○ SDCommands
- ○ RobotJ

SAFS\BIN Subdirectory

SAFS\BIN:
The SAFS\BIN directory contains executable VB Scripts (*.VBS) and CMD Batch files (*.BAT). The workhorse of these are the BATCH files. These will be used whenever manually performing Startup or Shutdown operations of various SAFS tools or engines. These BATCH files also serve as excellent examples of what commands are used to launch or shutdown various SAFS Framework tools.

In general, the VBScripts are safe to double-click irresponsibly. They are smart enough to warn you of what you are about to do. These scripts will also allow you to abort once you realize you don't know what you are doing  (Please abort any one you launch right now!)



The BATCH files are a different story. You double-click a BATCH file and it's off to the races whether you know what you're doing or not!  Your best defense is to view these in a text editor, but we will also be using them when exploring more of RRAFS and the SAFS Framework.

Take notice of the **XSL*.BAT** files in this BIN directory. These are the counterparts to the XSL*.XSL and XSL*.XML files reviewed in the KEYWORDS directory. These are the BATCH files that actually execute the transformation of the XML using the XSL.

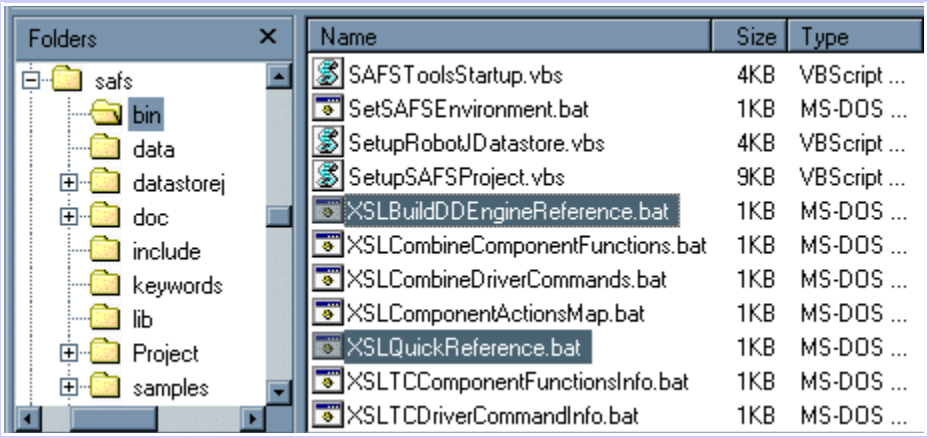
(cont'd)

SAFS\BIN Subdirectory (cont'd)

(cont'd)

The following BATCH files can build local copies of the SAFS Keyword Reference doc into your SAFS\DOC directory. Just double-click them in Windows Explorer!

1. **XSLBuildDDEngineReference.BAT**
2. **XSLQuickReference.BAT**



The remaining XSL*.BAT files are used to transform the data of the XML files and store it into the SAFS\DATA directory for various other purposes. You can use these as examples to build your own data transformations! Double-click each of these XSL*.BAT files and see what shows up in the SAFS\DATA directory!

After executing XSLComponentActionsMap.BAT, what file appears in SAFS\DATA?

SAFS\DATASTOREJ Subdirectory

SAFS\DATASTOREJ:
So, you're still with us, eh? Good for you!

The SAFS\DATASTOREJ directory is really nothing more than a template used to create and\or enable a Rational RobotJ\XDE Tester Datastore for use with the SAFS Framework. The files are either manually copied to the Datastore, or the VBScript **SetupRobotJDatastore.VBS** is copied from the SAFS\BIN directory and then executed.

We will go through all that later when we enable XDE Tester for execution.

Every file in this DatastoreJ template has what 10-letter root name?

Interlude: Studying Gravitational Anomalies...

For more than 10 years, scientists have been puzzled by measurements bounced off the Pioneer space probes intended to measure the Doppler shifts created by the two spacecraft. The Doppler shifts tell scientists how fast the probes are travelling out of the Solar System.

However, these measurements suggest the probe velocities are not as high as they should be. Something unexpected is, in effect, slowing them down. All tests and models attempting to account for this by the known laws of physics deduced over the past decades have failed.

While scientists look for heady reasons to explain this gravitational anomaly, there are some who believe the answer is rather obvious:

The Earth just sucks more than it did back then!

SAFS\DOC Subdirectory

Are you smiling? Please. Then continue.

SAFS\DOC:
This directory immediately following an install is fairly empty. Only some static images and HTML pages exist here. However, if you build any of the Reference documentation with the BATCH files in SAFS\BIN then all the SAFS Keywords Reference docs, the DDEngineReference doc, and even a Quick Reference doc might be here.

SAFS\DOC\INFO:
Some ugly reference material rarely fit for humans. The contents in this directory may vary from time to time. Review it if you have an extraordinary need to know EVERYTHING that exists in the SAFS install! Ignore it if you have enough other stuff to worry about.

SAFS\DOC\STATIC:
This actually IS a relatively important directory. This directory contains a copy of the static content normally found in SAFS\DOC. There may be occasions when you want to purge the SAFS\DOC directory. Go right ahead! Then copy the contents of this STATIC directory into SAFS\DOC and start anew.

SAFS\INCLUDE Subdirectory

SAFS\INCLUDE:
You can reference the headers and lib files used to build various DLLs here. We don't know if anyone has ever tried to import these, or if they even work for such imports. But they are there for you if you need them, or if you just want to see what is publicly available in the DLLs. It ain't pretty. Trust us.

SAFS\LIB Subdirectory

SAFS\LIB:
The LIB directory is where all the JAR files for various tools in the SAFS Framework reside. Many of these need to be in the system CLASSPATH for proper execution. Some do not. The names of the JARs intuitively suggest their function.

If you really care, you can see what is inside each JAR -- including sourcecode -- by using the Java JAR program, WINZIP, or some similar tool.

At over 1MB in size, which JAR file is the primary JAR file for SAFS?

SAFS\PROJECT Subdirectory

SAFS\PROJECT:
We've almost finished our not-so-whirlwind tour of a SAFS installation!

Like the DatastoreJ directory, **SAFS\PROJECT** is a template directory. However, this time we are talking about a generic template used for multiple testing tools -- a true SAFS project template. We use these for Rational projects and we use these for independent SAFSDRIVER projects.

When it comes time to create a new project we will copy this entire directory structure to the desired location. Alternatively, the **SetupSAFSProject.VBS** script in SAFS\BIN can be copied to the desired location and then executed instead. Hopefully, both result in the same thing!

runtTIDTest.bat
One interesting feature of this project template is that it has a ready-to-run test used to validate many SAFS features. With everything setup as expected, a quick double-click of this batch file will run a simple test. You can check the results of this test by looking in the Datapool\Logs subdirectory for the TIDTest.txt log file.

You can review the assets of the TIDTest test in the PROJECT and PROJECT\Datapool directories to see what the simplest of SAFSDRIVER tests and test configuration files might look like.

What is the name of the Rational INI file found in PROJECT\Datapool\Runtime?

SAFS\SAMPLES Subdirectory

SAFS\SAMPLES:
This is actually a place where many new users come and have a peek.

Batch: This subdirectory contains Batch files that cannot be used 'as-is'. They require some user-specific or project-specific editing before they will work properly. Once edited, these can be placed in SAFS\BIN with the rest of the Batch files or wherever it would be appropriate. We will address these later as needed.

Classics: This subdirectory actually has 2 separate samples. An old Notepad example using simple text assets and a "Classics Online" example using Microsoft Excel test assets.

Both of these examples are geared for Rational Robot (RRAFS) users. Although the Notepad example could easily be converted for independent use.

Java: The Java sample is a bit more substantial. The **swingapp.jar** can be executed directly by java with the following command-line:

java -jar swingapp.jar

Again, the sample Script is setup to execute from Rational Robot. This application is used to regression test the Java support for SAFS Engines.

Log Transforms: As the name suggests, there should be some sample XSL stylesheets for transforming XML logs to other formats. Use these for reference if you plan to make your own log summaries, timing reports, database imports, or for any reason you need to reformat the XML log into something else.

SAFS\SOURCE Subdirectory

Finally, the last directory in our Anatomy of SAFS!

SAFS\Source:
And there is nothing to see here! All the source for the Java-based SAFS Framework can be found in the JAR files in **SAFS\LIB**. If ever you want to extract the source out of the JAR files (like with WINZIP) then this is a convenient place to put it.

Break time: What is your favorite at-work beverage?

- ○ Water
- ○ Tea
- ○ Soda
- ○ Coffee

SQABasic Script Mods for SAFS

If you have already been using RRAFS for some time, then you might have some tried and true scripts already working for you. These older scripts will need to have a few lines added to bridge the new SAFS tools as seemlessly as possible into your familiar test environment.

If you are starting fresh and have no such test scripts, then skip this page and continue to the next one.

In Windows Explorer navigate to your **SAFS\Samples\Java\Script** directory and open the **JavaTest2.REC** file. Your test script needs to incorporate some of the function calls seen in this script. This sample script shows you where the functions should be added.

As seen in JavaTest2, add to the top of Sub Main:

```
DDGClearAppMapCache  (recommended)
DDVClearAllVariables      (optional)
```

As seen in JavaTest2, change any CloseTextLog and CloseXMLLog calls to one:

```
CloseAllLogs  MainLog, 1    [MainLog, or the log(s) your script uses]
DoEvents
```

If XML logging was done then add **required** XML header and footer:
(This should NOT be called if XML logging is not performed.)

```
status = LUCapXMLLog (MainLog.xmllog)
if status then SQAConsoleWrite "**** Error in XML Log Header or Footer! ****"
```

Finally, add the following code that will shutdown the SAFS Framework engines and tools at the end of your test. These can be called whether the SAFS Framework pieces are actually running or not:

```
SAFSShutdownDriverCommands
SAFSShutdownRobotJ
DelayFor 4000   'give these time to complete
SAFSShutdownSTAF
```

Note: ENGINES like SAFS/DriverCommands and SAFS/RobotJ **MUST** be shutdown **BEFORE** we shutdown STAF. Remember this during any manual shutdown operations you may attempt.

New SQABasic Scripts For SAFS

There are 3 sample test scripts provided in **SAFS\Samples**. Use these as templates for your own test scripts.

1. (Cycle) CycleDriverTest.REC in Samples\Classics\ClassicsC_V2001.ZIP
2. (Suite) JavaTest2.REC in Samples\Java\Script
3. (Step) DDNotepad.REC in Samples\Classics

They are all basically the same. The differences are, of course, which test table is called to start the test, and what log is written for the test.

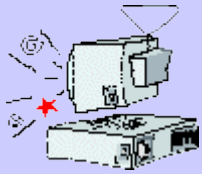
CycleDriverTest starts with a cycle table (Regression.CDD), JavaTest2 starts with a suite table (JavaTestSuite.STD), and DDNotepad starts with a step table (NotepadStepFile.TXT). Note, the use of a non-standard extension instead of .SDD for NotepadStepFile.TXT. As you can see, that is allowed.

At which level you start your test is largely a test design and development decision. But most tests start at the Cycle level.

Notice that CycleDriverTest.REC contains code for exporting test tables out of Microsoft Excel thru a separate script. That script is included in the ZIP file. If you plan to develop your test tables in Excel, then you will want to refer to these scripts and the ExcelUtilities library to export your tests to the required text format.

ExcelUtilities
http://safsdev.sourceforge.net/sqabasic2000/ExcelUtilities.htm

User-Abort Test {F11} Considerations



We know that you might find this hard to believe, but sometimes things don't always go as planned. The test acts funny, it fails for no apparent reason, or you forgot the app needed to be in a certain state. You smack that F11 function key to abort the test in progress and you think all is forgiven. But is it?

If SAFS tools or engines are running, there are a few more steps necessary to fully reset a test. This is especially true when logging into SAFSLOGS.

You started the test and initialized the logs. But when you abort the test you force the script to skip past the code that closes the logs. SAFSLOGS is still sitting there with open logs. And you will NOT be able to reinitialize those same logs again until they are closed.

Remember, when you abort the test you skipped past these important functions:

CloseAllLogs
SAFSShutdownDriverCommands
SAFSShutdownRobotJ
SAFSShutdownSTAF

The next page will review the SAFS\BIN assets available if you need to manually perform these functions.

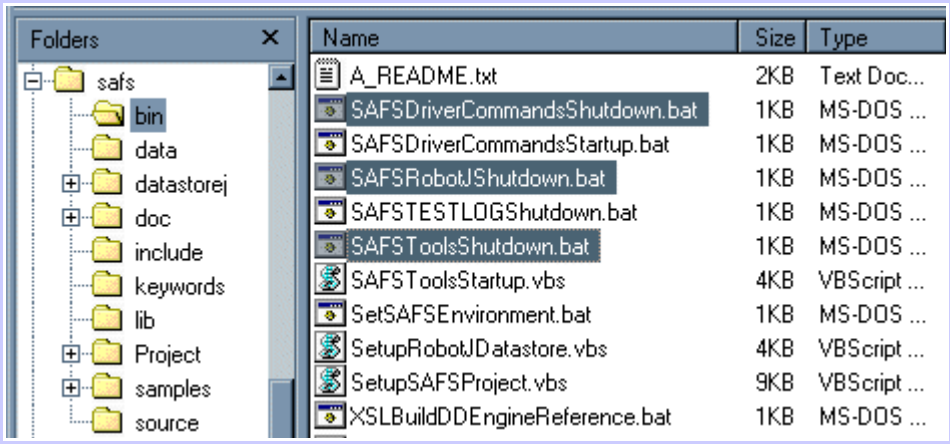
You could also make a separate script that simply calls those last 3 functions and you can reset everything in one quick script execution.

Review Manual Shutdown Scripts and Scenarios

Go ahead, navigate your Windows Explorer to the **SAFS\BIN** directory once again and review the batch files available to manually perform these shutdowns.

In SAFS\BIN you have:

SAFSDriverCommandsShutdown.BAT
SAFSRobotJShutdown.BAT
SAFSToolsShutdown.BAT



Now, SAFSToolsShutdown doesn't shutdown STAF, but it is good enough to shutdown SAFS services and close all open logs. That is what you are mostly concerned about.

When STAF was installed it also installed the "Shutdown STAF" shortcut in your Start Menu. Once SAFSDriverCommandsShutdown and SAFSRobotJShutdown have occured you can Shutdown STAF if you wish.

Ominous Warning:
If you attempt to Shutdown STAF *before* the DriverCommands and RobotJ engines are shutdown, you will find that the intertwined processes might not allow this and a significant timeout may have to expire. You may also have difficulty restarting STAF until the processes expire themselves. You may have to logoff and logon the system to force this, too.

So try to remember: shutdown DriverCommands and RobotJ first before you try to shutdown STAF. But hopefully your need to manually do this will be few and far between!

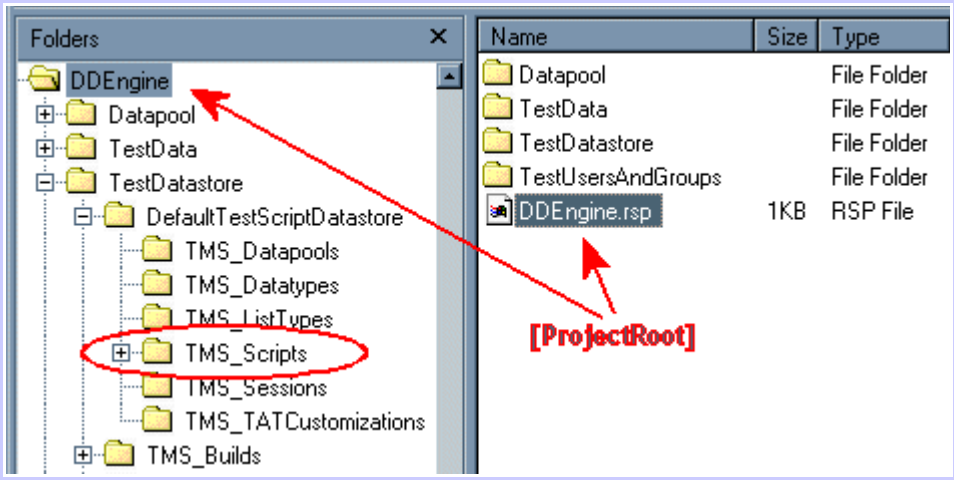
Prepare RRAFS Sample Test (No SAFS)

We are ready to run a simple test with Rational Robot. We are going to save the more complex process of enabling a specific Rational Project Repository for another lesson. Instead, we are going to run the exact same test used to verify the SAFS installation -- the one in SAFS\PROJECT.

Use any Robot project repository you already have. If you don't have a Robot project, then you must create one with the Rational Administrator. Once you are ready, proceed.

Copy the **runTIDTest.REC** script from **SAFS\Project** to your Rational project's TMS_Scripts directory. Historically, the path to TMS_Scripts in a Rational repository is:

[ProjectRoot]\TestDatastore\DefaultTestScriptDatastore\TMS_Scripts



After the script is copied into the TMS_Scripts directory, we must make it known to Robot. Launch Rational Robot... and wait ☺ ...then import the script with Robot's File Menu:

File->New->Script

Type in the GUI script name exactly: **runTIDTest** then press {ENTER}.

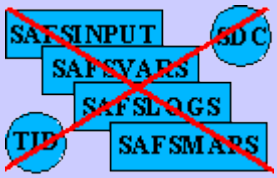
Robot should now display the script. If that didn't work, just copy-and-paste the contents of our runTIDTest.REC file into whatever new script was created.

Finally, compile that bad boy and let's see if we can run our test! 😂

Run RRAFS Sample Test (No SAFS)

If you made it to here, we'll assume you successfully imported and compiled the runTIDTest script in Rational Robot. So it's now time to run it. Before you do, here is what you can expect:

Not much.



If you went through the SAFS101 tutorial you may remember that this is an extremely simple test. The script is intended to verify the SAFS Framework pieces are installed and run correctly. However, this Robot script is NOT going to launch those SAFS Framework pieces because we have not "enabled" them for Robot just yet. Consequently, this test is expected to show some failures and warnings. So don't be alarmed when it does!

OK. "Play" the **runTIDTest** GUI script in Robot, wait for it to complete, and then proceed to the next page.

Note Test Log Features (No SAFS)

Hopefully, congratulations are in order! You ran a simple test and you should have a **TIDTest.Robot.txt** log displayed in Notepad. If you were actually too lazy to run the test, or it just didn't go as planned, open this test log for reference.

The actual log file is found in the **SAFS\Project\Datapool\Logs** directory. Note at the top of the log you can see this log was specifically written by Rational Robot. It starts with "SQARobot Test Log". There is also a WARNING and a FAILURE in the log because we have intentionally attempted a command that is not supported by the Rational Robot implementation. This command will become available AFTER we enable the SAFS Framework tools.

Finally, note the Status report at the end of the log. It lists the final tally of records and tests, passes and failures. There is a lot of whitespace surrounding that status report. This is only significant when we later compare this log to the one written by SAFS.

Reference Test Log:

```
================================================================================
SQARobot Test Log: c:\safs\project\datapool\Logs\TIDTest.txt
Version 1.1
Log OPENED 10-20-2004 22:16:34
  ------  START DATATABLE: c:\safs\project\datapool\TIDTest.cdd
          Application Map set to>c:\safs\project\datapool\TIDTest.MAP
- WARNING Unknown Driver Command in table c:\safs\project\datapool\TIDTest.cdd at Line  3
          C, Concatenate,"abc","def", "result"
          OnNotEqualGotoBlockID test values did not Match. Attempting branch to "ERROR"
          TRANSFERRING EXECUTION TO BLOCKID "ERROR" in table c:\safs\project\datapool\TIDTest.cdd at line 10
          Begin Block "ERROR"
**FAILED**Error in one or more SAFS modules!

  ------  END DATATABLE: CYCLE TABLE: c:\safs\project\datapool\TIDTest.cdd
 ======== BEGIN STATUS REPORT: c:\safs\project\datapool\TIDTest.cdd
             TOTAL RECORDS: 5
           SKIPPED RECORDS: 0

              TEST RECORDS: 1
             TEST FAILURES: 1
             TEST WARNINGS: 0
              TESTS PASSED: 0

          GENERAL FAILURES: 0
          GENERAL WARNINGS: 1
                IO FAILURES: 0
 ======== END STATUS REPORT: c:\safs\project\datapool\TIDTest.cdd

Log CLOSED 10-20-2004 22:16:34
```

Project RRAFS.INI

RRAFS provides the automator with the ability to configure various features of the framework. This is done with the **RRAFS.INI** file. There is a 'global' RRAFS.INI file in the **\Rational\Rational Test\sqabas32** directory where you installed the Rational tools and the RRAFS framework. Settings made or changed in this RRAFS.INI file affect all projects accessed by this machine.

We can override those global settings by placing a separate RRAFS.INI file in each project's **Datapool\Runtime** directory. Settings not present in the project's RRAFS.INI file will look-thru to whatever exists in the global RRAFS.INI file.

Go ahead and open the RRAFS.INI file located in the **SAFS\Project\Datapool\Runtime** directory. It should look just like this one.

Review the information in this file at your leisure and then proceed to the next page.

Sample RRAFS.INI Filename
http://safsworks.com/online/rrafs101/RRAFS.INI.txt

Enabling Basic SAFS Tools with AUTOLAUNCH

You should open the **RRAFS.INI** file from **SAFS\Project\Datapool\Runtime** in your favorite text editor. We are finally going to enable the SAFS Framework!

Note the semicolon used for comments. Items in a section that are optional will generally be commented out. Don't uncomment any of these items unless you know what they are for, why you are doing it, and what valid values you can use.

Bypass the information on [DIRECTORIES], we aren't going to deal with that right now. Go and set **AUTOLAUNCH=TRUE** for each of the following sections in that RRAFS.INI:

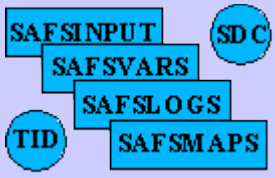
[SAFSVARS]
[SAFSMAPS]
[SAFSLOGS]
[SAFS_DriverCommands]

You must *NOT* set AUTOLAUNCH=TRUE for SAFS/ROBOTJ at this time. We will deal with that later, too.

Once you have made these changes, save them and move on to the next page where we will execute our test with the SAFS Framework turned on!

Run RRAFS Sample Test WITH SAFS

With the SAFS Framework enabled we are going to run that very same **runTIDTest** GUI script in Robot. This time we can expect to see a little more action when the test executes.



With SAFS enabled, Rational Robot is going to turn over several functions to SAFS that Robot would normally handle itself. This includes the management of Application Maps, global variables, and logging. We are also going to have access to the Java-based SAFS/DriverCommands engine and all the new Driver Commands it provides.

When we play the runTIDTest script we should see Robot automatically launch STAF, the SAFS tools, and the DriverCommands engine. Generally, these will appear in DOS or CMD windows that may or may not become minimized when they run. When the test is completed, these tools should automatically shutdown and all these CMD windows will go away.

Go ahead and "Play" the **runTIDTest** GUI script in Robot. Continue on to the next page when the test is complete.

Compare Test Log Features

Refer to the **TIDTest.Robot.txt** log that was just created in the **SAFS\Project\Datapool\Logs** directory, or to this one we have stored. Compare that with this one from the previous run when SAFS was not enabled.

Carefully review the two log similarities and differences. The new log shows it is NOT a Robot created log. Note the UNIMPLEMENTED command WARNING and the test FAILURE are gone in the new log, too.

Finally, notice that the Status Report at the end of the new log has each line prefixed with 'REPORT'. These are all indicators that show we successfully ran the test with the SAFS Framework properly enabled.

Congratulations! Now let us do a final review of why this is a good thing...

Sample Test Log:

```
Version 1.1
Log OPENED 10-20-2004 22:24:56
  ------  START DATATABLE: c:\safs\project\datapool\TIDTest.cdd
          Application Map set to>c:\safs\project\datapool\TIDTest.MAP
          Concatenate successful., abcdef
          OnNotEqualGotoBlockID did not branch.  Provided values did match.
          "abcdef"="abcdef"
    OK    TIDTest Services all present and working.
  ------  END DATATABLE: CYCLE TABLE: c:\safs\project\datapool\TIDTest.cdd
REPORT     BEGIN STATUS: c:\safs\project\datapool\TIDTest.cdd
REPORT    TOTAL RECORDS: 5
REPORT  SKIPPED RECORDS: 0

REPORT     TEST RECORDS: 1
REPORT    TEST FAILURES: 0
REPORT    TEST WARNINGS: 0
REPORT     TESTS PASSED: 1

REPORT GENERAL FAILURES: 0
REPORT GENERAL WARNINGS: 0
REPORT      IO FAILURES: 0
REPORT       END STATUS: c:\safs\project\datapool\TIDTest.cdd

Log CLOSED 10-20-2004 22:24:58
```

Review What You Have Gained

This has been a long lesson, but hopefully you have gained a good understanding of how to enable the SAFS Framework, and how to deal with any problems that may arise down the road.

With SAFS enabled, Rational Robot gains access to a growing list of commands, engines, and features without having to add code into the Robot libraries themselves.

Open the SAFS Keyword Reference and review the many Driver Commands in the reference. There are 8 or more different types of Driver Command categories listed. Go thru each one and take note of which engines support the various commands. When you enable SAFS, you now have access to all the commands from the SAFS/DriverCommands engine. Those are listed with the SDC icon.

In addition, with SAFS enabled we can turn to enabling XDE Tester for those that have both Robot and XDE Tester licenses. Plus, any open source engines that become available will be accessible to Rational Robot, too!

SAFS Keyword Reference
http://safsdev.sourceforge.net/sqabasic2000/RRAFSReference.htm

Quiz: RRAFS and the SAFS Framework

This quiz should only be attemped AFTER you have completed the accompanying lesson on Enabling SAFS.

1 1 Marks

Whether you are using Rational Robot, WinRunner, or any other testing tool;

What does SAFS offer?

Answer:
- a. All of these
- b. Access to XDE Tester scripts and commands
- c. More Driver Commands via SAFS/DriverCommands (SDC)
- d. Access to future open source engines and tools
- e. Access to shared data and tools

2 1 Marks

What is the default directory for 'ready-to-run' SAFS BATch files that can start or stop various SAFS Framework tools for ANY project?

Answer:
- a. C:\SAFS\BIN
- b. .\Rational\Rational Test\sqabas32
- c. C:\SAFS\Samples\Batch

3 1 Marks
What is the default directory containing SAFS BATch templates that should be copied and modified for each unique project repository?

Answer:
- a. C:\SAFS\Samples\Batch
- b. .\Rational\Rational Test\sqabas32
- c. C:\SAFS\BIN

4 1 Marks
If you find this necessary, what batch script in the default SAFS install directory would be used to shutdown the SAFS/DriverCommands engine?

Answer:
- a. C:\SAFS\BIN\SAFSToolsShutdown.BAT
- b. C:\SAFS\Samples\Batch\ShutdownSAFSDriverCommands.BAT
- c. C:\SAFS\BIN\SAFSDriverCommandsShutdown.BAT

5 1 Marks
If you have to shutdown SAFS manually, what is the **required** shutdown sequence?

Answer:
- a. Shutdown all services, then engines, and then STAF.
- b. Shutdown all 'engines' and then shutdown STAF.
- c. Shutdown STAF and it will shutdown all engines and services.

6 1 Marks
What command can be issued at any command prompt to list all the SAFS tools currently running as services in STAF?

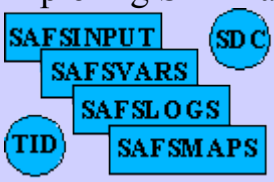
Answer:

7 1 Marks
What command entered at any command prompt will list all running STAF client 'handles' -- both services (tools) and engines?

Answer:

Lesson

7 Exploring STAF and the SAFS Services



RRAFS will automatically use the SAFS services when they are enabled. But that doesn't have to be the end of that. You can take advantage of STAF and the SAFS services within your own scripts, with other tools, and for many things not related to RRAFS. These tools are truly independent of RRAFS!

The "Remote Viewing" Program

Explore STAF and SAFS Services

Quiz: STAF and SAFS Services

The "Remote Viewing" Program

The problem:
We want to perform multi-user testing from multiple machines running full application clients -- not virtual-users that simply mimic the application client. We have full Rational installs on each client machine and each client machine already uses a standard Rational Robot license.

A synchronized multi-user test using multiple Rational Robot machines traditionally would use TestManager to direct the tests across all machines. But this type of test management requires VU licenses at a cost over and above the cost of the Robot licenses we have already purchased and intend to use.

The solution:
Each client machine already uses STAF and our SAFS services for testing. STAF is also very capable at communicating to remote machines in a distributed environment. Instead of using TestManager to initiate and control RRAFS tests we can use STAF.

If necessary, the controlling machine can transfer all test assets to the remote clients. We can also initiate tests on each client machine whenever we like. Each machine maintains its own separate RRAFS log and these can later be merged if desired.

STAF allows us to connect to other STAF-enabled machines, monitor test information and data executing on the remote machine, even remotely manipulate data that can affect or alter the execution of remote tests.

Explore STAF and SAFS Services

RRAFS and STAF and SAFS, Oh My!

We've installed RRAFS and played with some tests that run by themselves. That is great! But what about this STAF stuff we keep hearing about? And the SAFS Framework? What is the difference? We have it, but what the heck does it all do?

Let's find out!

We'll do a quick review of STAF. Most of the background info for STAF will have to wait for a more advanced tutorial. But you will be getting a good taste of using STAF.

After that we will do a quick hands-on review of the SAFS Services. This will also prep one of our C:\SAFS\Sample\Batch files for use in the next section.

Finally, we will do more extensive interaction with SAFS Services through the command line. This should show you how these tools and services can be used by any process or program running on the machine -- not just Rational Robot!

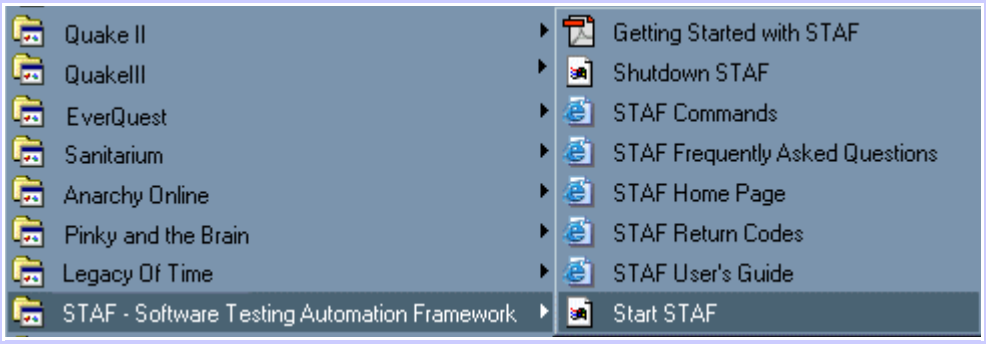
Review STAF

Software Testing Automation Framework

As cool as the SAFS Framework is, a great deal of the magic is provided by the bridge or middleware "glue" from another project on SourceForge: **STAF**

Optionally review the STAF Home Page and the STAF User's Guide.

Much of this information is available in the C:\STAF\docs directory where STAF was installed and in the Start Programs Menu for STAF. The installed version of the doc is more accurate because it will match the version of STAF that is installed.

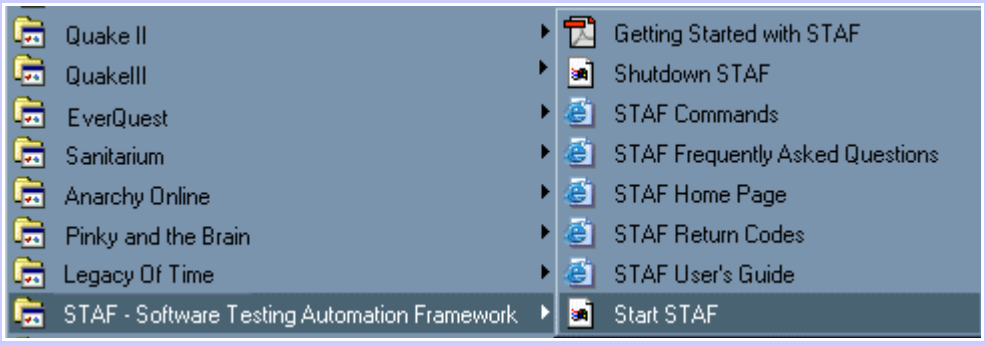


After you have had enough fun reading that, we can proceed.

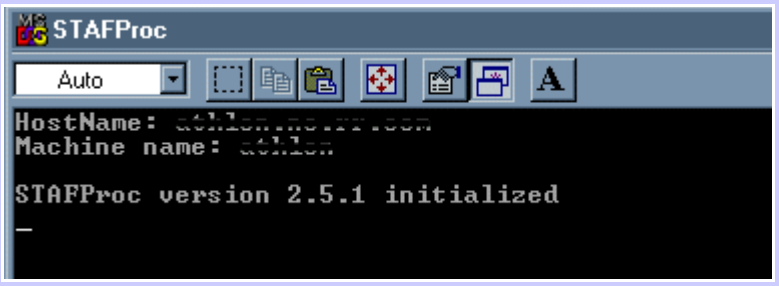
Starting STAF Manually

If you read the STAF User's Guide you would know that STAF has many interfaces. RRAFS talks with STAF via C, VB, and Java. There are also interfaces for Perl, Python, and others. Let's explore manually playing with STAF through the command line.

Start STAF via the STAF Program Menu

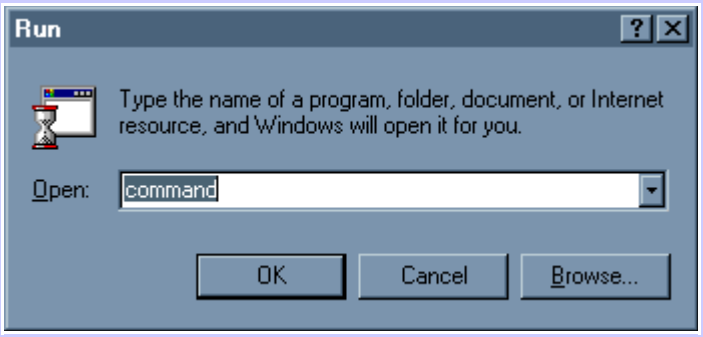


A Command Window should appear showing STAF has started successfully.



Launch a Command Window

Launch a Command Window via Start->Run



Type "command" and hit ENTER or the OK button to launch the command window. It is OK that yours will open in a different directory than the one shown below.

You are about to play with STAF manually. While this is really cool, most testers will not ever have to actually do this. So if this begins to feel overwhelming, don't worry about it. You may not ever have to actually do this on-the-job.

Now let's proceed to see some cool STAF stuff!

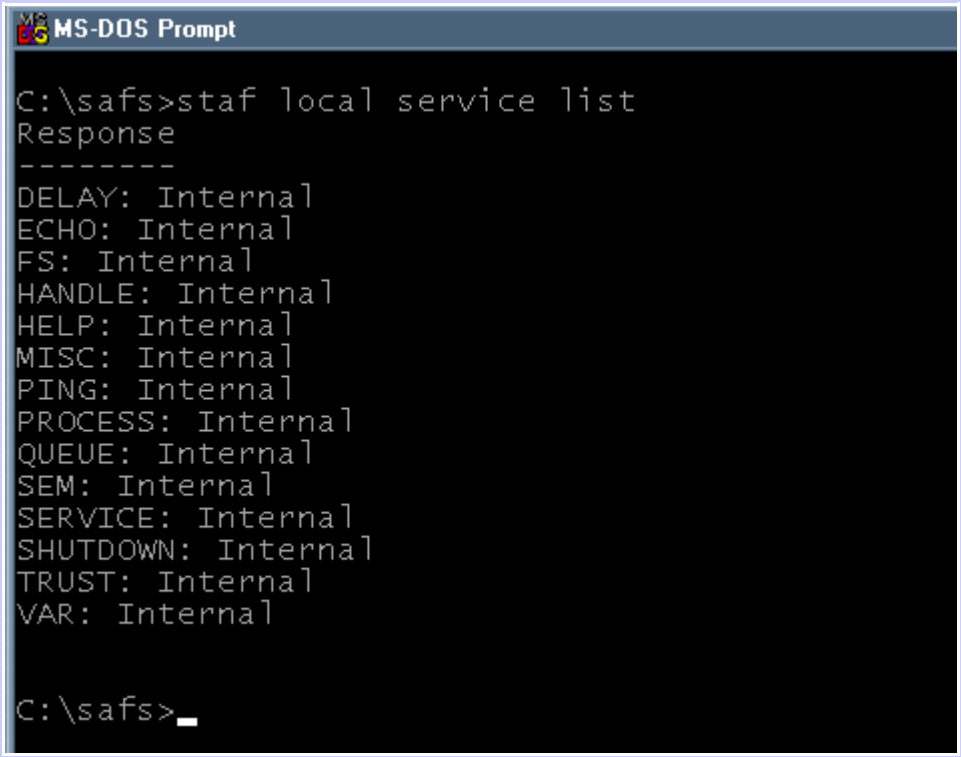
Issuing Simple STAF Commands

So lets begin with some basics. STAF can talk to other STAF clients on remote machines. But right now we are only interested in talking with STAF on our local machine. Thus all our commands will begin with "staf local".

We'll start by asking the local STAF service named "SERVICE" to list all services that are currently running. (We know, that's just too much 'service' to deal with!

In the command window, type the following command and press ENTER:

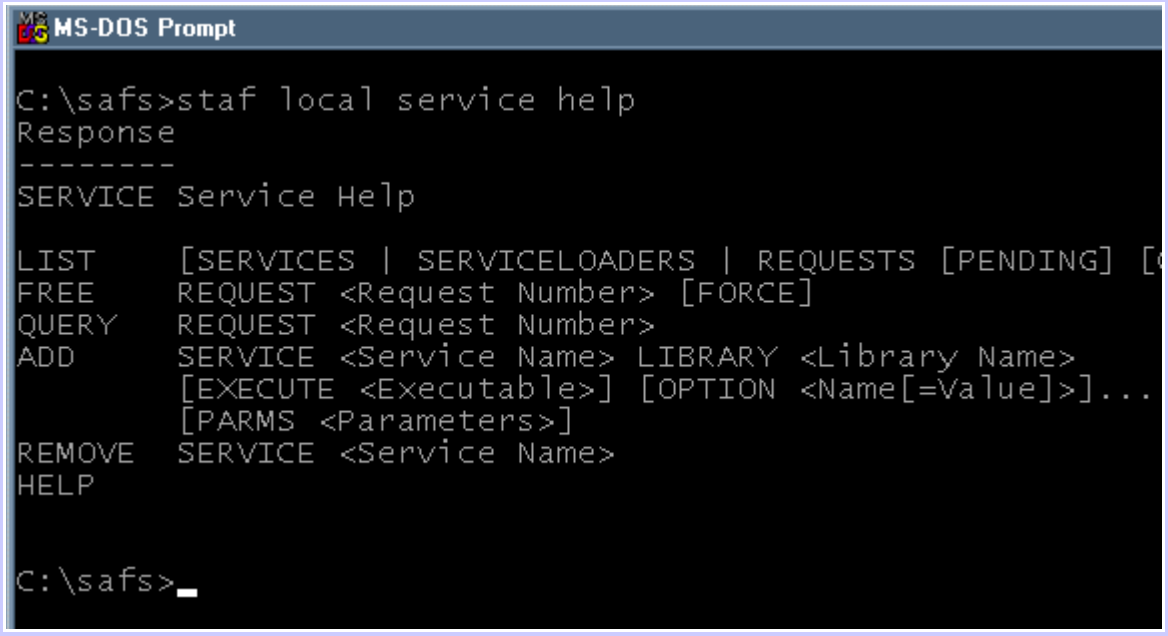
staf local service list



Getting HELP for a STAF Service

Most services support the "help" command. This typically tells you all the commands available via the service. In the command window type the following command and press ENTER to view the HELP info for the SERVICE service:

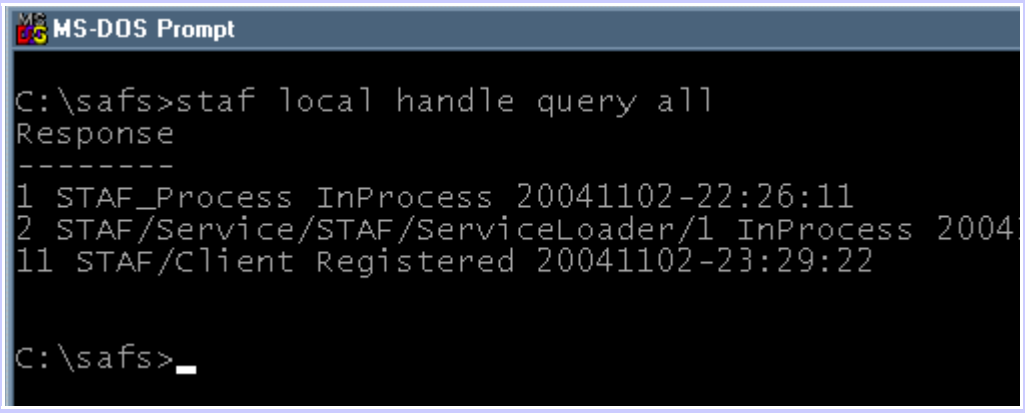
staf local service help



Getting STAF Client Handles

Another STAF service, the HANDLE service, tells us the handle IDs for clients registered with STAF. This includes clients that are NOT services, like our STAF command line prompt! Enter the command below and press ENTER:

staf local handle query all



Our STAF command line window shows up as "STAF/Client" and its handle increments every time we use it because we start and stop the command line client with each command we enter. And every client that registers with STAF gets a new unique HANDLE ID.

Browse STAF Services

Go ahead and review the internal STAF service offerings by using the various commands:

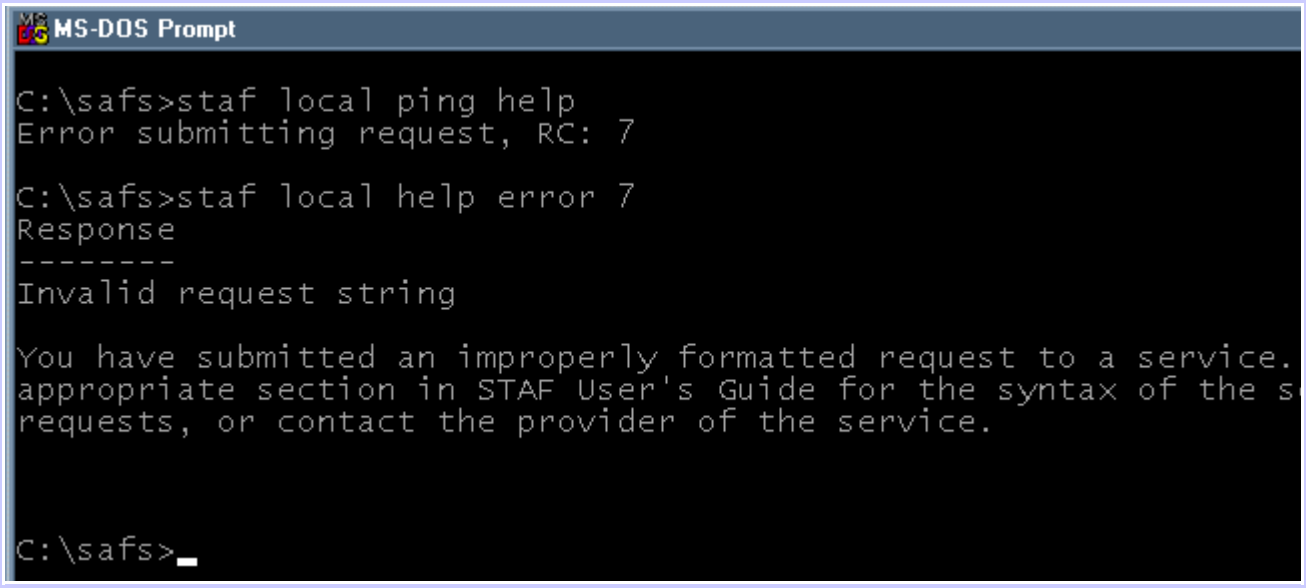
staf local service list
staf local <servicename> help

If you encounter any STAF errors you can often find out what the return code (RC) means with help from the HELP service:

staf local help error <error number>

For example, the PING service does not support "help" in STAF 2.5.1:

staf local ping help
staf local help error 7



Shutdown STAF Manually

There is a "Shutdown STAF" item in the STAF Program Menu. But we can also shutdown STAF by issuing "shutdown" to the SHUTDOWN service in STAF.

Type the following command into the command window and press ENTER:

staf local shutdown shutdown

The command window in which STAF was launched should signal it is proceeding with a "normal" shutdown. If your command windows are configured to do so, then that STAF command window will completely close.

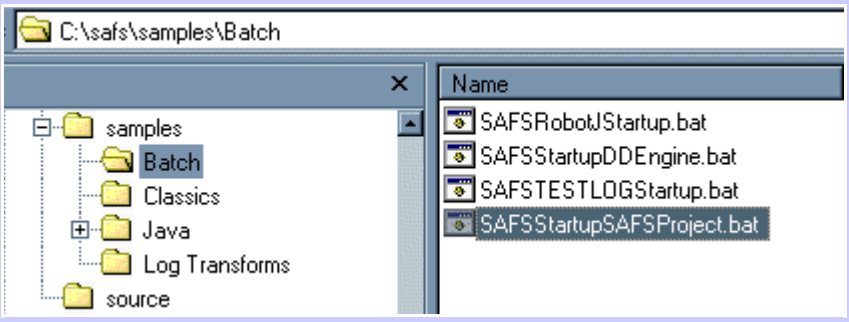
Well, there you go! Hopefully you met with success while playing with STAF. If it seems a bit overwhelming, don't worry about it. Most testers will never have to play with STAF like this since RRAFS tries to make it fairly transparent and automatic.

Next we begin to review SAFS-specific services in "Exploring SAFS Services"

Prepare Sample Batch File

Now that you are completely and utterly familiar with STAF, we will move on to reviewing the SAFS Services directly. To make this simpler, we need to modify one of our sample batch files.

Open **SAFSStartupSAFSProject.BAT** from the SAFS\Samples\Batch directory in your favorite text editor:



Review and correct any path information in this file. Note the first set of directories on the left may be correct if you installed to the default SAFS directory. If you have to edit this first set, note the Unix file separator is used instead of the standard Windows file separator.

```
execute c:/safs/lib/safsmaps.jar PARMS dir c:\safsproject\datapool
execute c:/safs/lib/safsvars.jar
execute c:/safs/lib/safsinput.jar PARMS dir c:\safsproject\datapool
execute c:/safs/lib/safslogs.jar PARMS dir c:\safsproject\datapool\logs
```

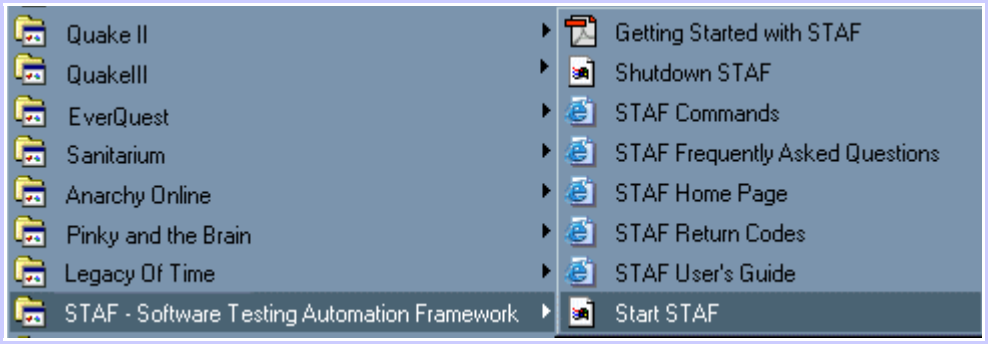
Note, the set of directories on the right may need editing. In this snapshot the directories on the right more than likely need to be corrected ("safs\project" instead of "safsproject"). Later releases might have this 'fixed' for you already. Oddly enough, this right side set uses the Windows file separator!

Review the commands themselves and you see how the SAFS services can be launched from the command line. **Copy this corrected file to the SAFS\BIN directory** where all the other batch files are located.

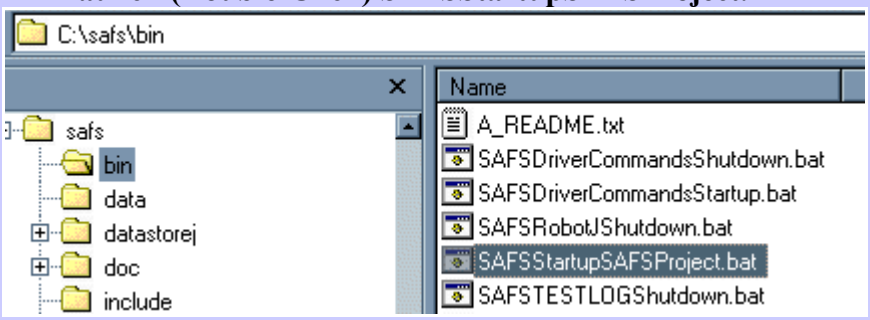
Starting STAF Manually (Again)

Now lets get back to playing! Launch STAF, then launch the SAFSStartupSAFSProject.BAT file we just copied to SAFS\Bin. STAF must be launched first before anything else STAF-related will function.

Start STAF via the STAF Program Menu



Launch (Double-Click) SAFSStartupSAFSProject.BAT



If your console window is configured to remain open you should see that the RESPONSE string for the invocation of each service is blank (no errors). Now let's go see what these SAFS services can do for us!

Review SAFS Services List

Launch a Command Window via Start->Run

Let's make sure our services are actually available to us.
In the command window type the following command and press ENTER.

staf local service list

```
PING: Internal
PROCESS: Internal
QUEUE: Internal
SAFSINPUT: External library jstaf, exec c:/safs/lib/safsinput.jar
SAFSLOGS: External library jstaf, exec c:/safs/lib/safslogs.jar
SAFSLOGSLOG: External library STAFLog
SAFSMAPS: External library jstaf, exec c:/safs/lib/safsmaps.jar
SAFSVARS: External library jstaf, exec c:/safs/lib/safsvars.jar
SEM: Internal
SERVICE: Internal
```

Review the list of running services. Our SAFS services should now be among the list of 'Internal' services you saw previously.

Review SAFS Services Help

Let's now review the HELP provided by each of the SAFS services. Enter each of these commands one at a time and review the HELP response from each service:

staf local service safsvars help
staf local service safsmaps help
staf local service safslogs help
staf local service safsinput help

Remember, you can find out information about errors with:

staf local help error <error #>
Ex: staf local help error 7

You can find more detail in the online documentation available for each service:

SAFSVARS JavaDoc
SAFSMAPS JavaDoc
SAFSLOGS JavaDoc
SAFSINPUT JavaDoc

Once you have finished reviewing these, shutdown STAF from the command line:

staf local shutdown shutdown

This will automatically shutdown our SAFS services, too.

SAFSVARS JavaDoc
http://safsdev.sourceforge.net/doc/org/safs/staf/service/SAFSVariableService.html

SAFSMAPS JavaDoc
http://safsdev.sourceforge.net/doc/org/safs/staf/service/SAFSAppMapService.html

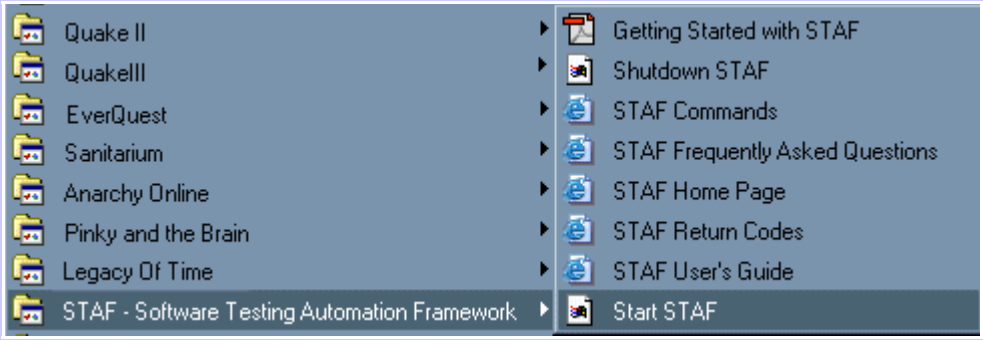
SAFSLOGS JavaDoc
http://safsdev.sourceforge.net/doc/org/safs/staf/service/logging/SAFSLoggingService.html

SAFSINPUT JavaDoc
http://safsdev.sourceforge.net/doc/org/safs/staf/service/SAFSInputService.html

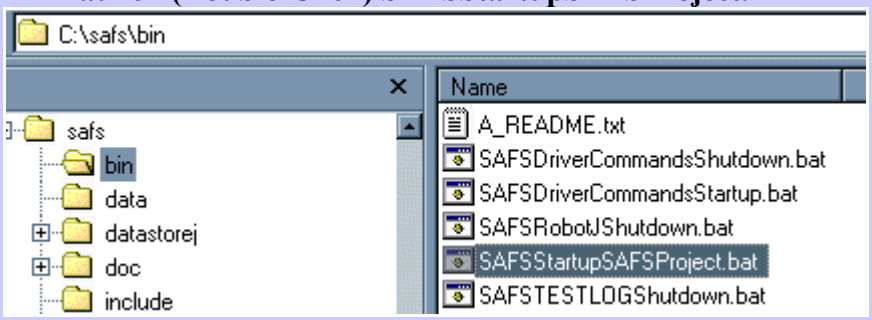
Launch STAF and SAFS Services (Again)

Now lets get back to playing! Launch STAF, then launch the SAFSStartupSAFSProject.BAT file in SAFS\Bin. Remember, STAF must be launched first before anything else STAF-related will function.

Start STAF via the STAF Program Menu



Launch (Double-Click) SAFSStartupSAFSProject.BAT



If your console window is configured to remain open you should see that the RESPONSE string for the invocation of each service is blank (no errors). Now let's actually play with the SAFS Services!

Launch a Command Window and Check Service List

Launch a Command Window via Start->Run

Let's make sure SAFS services are actually available to us.
In the command window type the following command and press ENTER.

staf local service list

```
PING: Internal
PROCESS: Internal
QUEUE: Internal
SAFSINPUT: External library jstaf, exec c:/safs/lib/safsinput.jar
SAFSLOGS: External library jstaf, exec c:/safs/lib/safslogs.jar
SAFSLOGSLOG: External library STAFLog
SAFSMAPS: External library jstaf, exec c:/safs/lib/safsmaps.jar
SAFSVARS: External library jstaf, exec c:/safs/lib/safsvars.jar
SEM: Internal
SERVICE: Internal
```

Review the list of running services. Our SAFS services should again be among the list of 'Internal' services.

OPEN App Map 'TIDTest.MAP' in SAFSMAPS

In the command window, review the Help for the OPEN command in SAFSMAPS. Type the following command and press ENTER:

staf local safsmaps help

```
SAFSAppMapService HELP

HANDLEID
OPEN     <appmapID> FILE <filename> [STORED] [MAPPED] [DEFAULTMAP]
SAFSVARS [<safsVarService>]
```

Now let's tell SAFSMAPS to open the TIDTest.MAP file used in the simple tests we ran earlier. Type the following command and press ENTER:

staf local safsmaps open tidmap file TIDTest.MAP

```
C:\safs>staf local safsmaps open tidmap file TIDTest.MAP
Response
--------
OPEN:tidmap:STORED:DEFAULTMAP:c:\safs\project\datapool\TIDTest.MAP

C:\safs>
```

For your own reference, you can open the TIDTest.MAP file from the listed directory into a text editor or viewer. The contents should be pretty basic like this below:

[ApplicationConstants]
Var1="abc"
Var2="def"

Review SAFSVARS Variables List

Now we are going to look at SAFSVARS and a pretty powerful interaction between SAFSVARS and SAFSMAPS.

Type the following command in the command window and press ENTER:

staf local safsvars help

```
SAFSVariablesService HELP

HANDLEID
SAFSMAPS [<safsAppMapService>]
GET <varname>
SET <varname> VALUE <value>
LIST
COUNT
RESET
DELETE <varname>
RESOLVE <string> [SEPARATOR <char>]
HELP
```

Notice the commands you can use to GET and SET variables. You can DELETE individual variables or clear out all variables with RESET.

Right now we are simply going to show that no variable values currently exist.

Type the command below and press ENTER. You should see that there are no variables currently listed.

staf local safsvars list

```
C:\safs>staf local safsvars list
Response
--------


C:\safs>
```

Even though no variables have been defined, we can still use the CONSTANTS stored in the App Map as if they were normal variables. To demonstrate this we are going to be grabbing a CONSTANT out of the App Map in the SAFSMAPS service through the SAFSVARS service!

Retrieve a SAFVARS Variable Value (or CONSTANT)

We are ready to see that magic right?! So let's do it!

Type the following command in the command window and press ENTER:

staf local safsvars get Var1

```
C:\safs>staf local safsvars get Var1
Response
--------
abc

C:\safs>
```

Notice the GET command wants only the variable name. It does not need the caret (^) or circumflex that we use in SAFS test tables.

As you should be able to see, the SAFSVARS service recognized it had no variable named 'Var1' in storage, so it sought a 'Var1' CONSTANT from the current App Map in the SAFSMAPS service! And as we saw previously, 'Var1' in our App Map has the value "abc". Pretty cool eh?!

Initialize a SAFSLOGS Log

Finally, lets try opening, writing to, and closing a centralized log. But first, review the SAFSLOGS Help by typing the following command and pressing ENTER:

staf local safslogs help

```
SAFSLoggingService HELP
HANDLEID
INIT        <facname> [TEXTLOG [<altname>]] [XMLLOG [<altname>]] [TOOLLOG]
            [CONSOLELOG] [ALL] [LINKEDFAC <name>] [OVERWRITE]
QUERY       <facname> [TEXTLOG | XMLLOG | TOOLLOG | CONSOLELOG | ALL]
LIST        [SETTINGS]
SUSPENDLOG  <facname> | ALL
RESUMELOG   <facname> | ALL
LOGLEVEL    <facname> [DEBUG | INFO | WARN | ERROR]
LOGMESSAGE  <facname> MESSAGE <msg> [DESCRIPTION <desc>] [MSGTYPE <msgType>]
CLOSE       <facname> | ALL
HELP        [MSGTYPE]
VERSION
```

Now initialize a new log. Type the following command and press ENTER:

staf local safslogs init mylog TEXTLOG OVERWRITE

Our log should now be open. We can verify this by issuing the LIST command to SAFSLOGS:

staf local safslogs list

```
C:\safs>staf local safslogs list
Response
--------
1;MYLOG;

C:\safs>
```

Logging into a SAFSLOGS Log

We've verified 'MyLog' has been opened, so let's send a message to it!

Type the following command in the command window and press ENTER:

staf local safslogs logmessage mylog message "Hello World!"

```
C:\safs>staf local safslogs logmessage mylog message "Hello World!"
Response
--------
TOOLLOG=false
CONSOLELOG=false
LOGLEVEL=2
```

What is all the info in the Response from logging our message? Important info for out testing tools!

When a tool like Robot or WinRunner logs a message the SAFSLOGS service tells the tool whether or not it should log the same message to its own logs, to any console it may have, and the level of logging currently enabled (0=ERROR, 1=WARN, 2=INFO(default), 3=DEBUG).

Close the log with the following command. You can then verify the log is closed with the LIST command again.

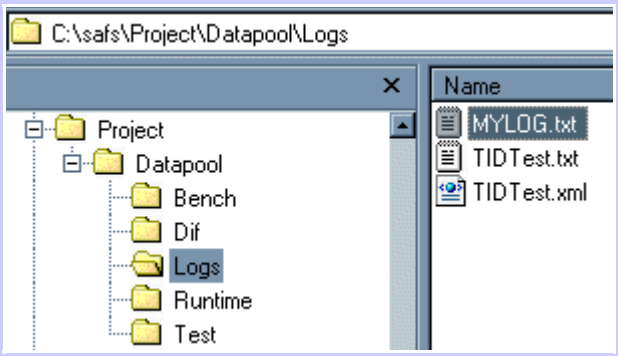
staf local safslogs close mylog
staf local safslogs list

```
C:\safs>staf local safslogs list
Response
--------
0;
```

Review the TextLog Output of SAFSLOGS

Now that we have closed our log we can open it and see what's inside!

Open '**MYLOG.TXT**' which should be found in our **SAFS\Project\Datapool\Logs** directory:



Note, you won't find the logs here -- not even partial logs -- until AFTER the log has been closed. SAFS does some processing on any Text and XML logs before it can write them to their final destinations. So always make sure you close any or all logs before you go hunting down their contents!

Open the log in your text editor or viewer and you should see your infamous message!

Version 1.1
Log OPENED 11-04-2004 23:08:17
Hello World!
Log CLOSED 11-04-2004 23:36:13

Well, there you go! Great job! Lets close down STAF and be done with this lesson.

staf local shutdown shutdown

Quiz: STAF and SAFS Services

RRAFS and SAFS and STAF, Oh My! Let's see what you can recommember about these new tool-independent offerings...

1 What does the acronym 'STAF' stand for?

1 Marks

Answer:

2 The SAFS Framework requires and bundles STAF. What is the default installation directory for STAF?

1 Marks

Answer:
- ○ a. C:\STAF
- ○ b. C:\Program Files\STAF
- ○ c. DDE_RUNTIME: .\Rational\Rational Test\sqabas32
- ○ d. C:\SAFS

3 The STAF User's Guide provides a wealth of info on using STAF. Besides being online at staf.sourceforge.net, what file in your STAF\docs directory also contains the STAF User's Guide?

1 Marks

Answer:
- ○ a. STAFFAQ.htm
- ○ b. STAFHome.htm
- ○ c. STAFUG.htm
- ○ d. STAFRC.htm

4 Which programming languages have the means to communicate with each other thru STAF?

1 Marks

Answer:
- ○ a. Perl
- ○ b. C/C++
- ○ c. Java
- ○ d. All of these and more!
- ○ e. Python
- ○ f. VB

5 1 Marks

What is the default installation directory for the independent SAFS Framework on Windows?

Answer:

- a. C:\SAFS
- b. C:\STAF
- c. C:\Program Files\SAFS
- d. DDE_RUNTIME: .\Rational\Rational Test\sqabas32

6 1 Marks

What command can be issued at any command prompt to list all the SAFS tools currently running as services in STAF?

Answer:

7 1 Marks

What command entered at any command prompt will list all running STAF client 'handles' -- both services (tools) and engines?

Answer:

8 1 Marks

Of the SAFS Services currently available, list the 3 that are used by RRAFS when the SAFS Framework is enabled.

Answer:

- a. SAFSMAPS
- b. SAFSINPUT
- c. SAFSVARS
- d. SAFSLOGS

9 1 Marks

We can get Help on the syntax supported by each SAFS service via online documentation and by a simple request issued to each service from the command-line. What STAF command to the local running SAFSMAPS service will show us the commands it supports?

Answer:

10 1 Marks

Most SAFS services provide a command that allows you to List all the current things they are tracking like open logs, open app maps, variable values, etc... The command is generally the same for all services. What is the command to get a list of all variable values from the local running SAFSVARS service?

Answer:

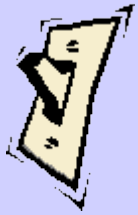
11
1 Marks

In addition to simply clicking a 'Shutdown STAF' Program Icon we can also shutdown all of STAF via a command to the STAF SHUTDOWN service. What is the command to issue to the local running SHUTDOWN service instructing STAF to shutdown?

Answer:

Lesson

8



Enabling Rational XDE Tester

If you are lucky enough to have both Rational Robot and Rational XDE Tester, then have we got news for you. SAFS allows us to use both tools at the same time in the same test! Robot and XDE Tester can 'talk' to each other and we can use the best features of both tools to get the job done.

Recovering From Change

Enabling Rational XDE Tester

Quiz: Enabling Rational XDE Tester

Recovering From Change

Those experienced with test automation and those who have heard the countless recounts of the stories know that an application in development will change. These changes can often impact the test automation already in place and require some amount of maintenance or rework to get things right again.

Those experienced with test automation also know that patches and upgrades to the testing tools themselves can also bring the same type of unwanted change! And the experience below is a true example.

We recently upgraded our suite of Rational tools from V2002 to V2003. There was a great deal of cool new stuff and other stuff that just started working more reliably than it did in V2002. But we also found some hidden surprises that could have killed our test automation suites if it were not for the SAFS Framework!

We found that a great number of our Web and Java tests were failing after the V2003 upgrade due to 'enhancements' in Rational Robot. Trees and Tables were not being handled the same way as in V2002. We also found that several bugs we had patched with a Robot HOTFIX for V2002 were not fixed in V2003. (And, of course, the V2002 HOTFIX could not be applied to V2003.)

We wanted to move forward with the upgrade, but that came at what could have been a huge impact to production testing. So we explored available options.

Thankfully, all our tests were RRAFS keyword-driven tests. While we had only used Rational Robot for execution up to this point, we decided it was time to see if Rational XDE Tester could be part of the solution!

We went through the tasks to enable Rational XDE Tester for the SAFS Framework and we were pleasantly amazed! In many situations XDE Tester ran the same test significantly faster. More importantly, the Tree navigation that was failing in Rational Robot was not a problem for XDE Tester! Yet XDE Tester was by no means going to be a complete replacement for Rational Robot. Like any tool, it had its own problems.

Recovering From Change (cont'd)

We found that there were some parts of the test that XDE Tester could not handle. For example, XDE Tester did not always 'see' our Java Popup menus. At other times they were not a problem. Rational Robot, on the other hand, was not giving us any problems at all with these same Java Popup menus.

The solution, of course, was that we would now use both tools. Each tool would handle the part of the test for which it was best suited allowing our daily production tests to continue while we worked the newfound issues with Rational support.

The SAFS Framework allowed us to introduce our tests unmodified to a new testing tool (XDE Tester) and we were able to circumvent the immediate Robot-induced test failures. Yet we could still use Rational Robot for those parts of the test that XDE Tester could not handle. There is an excellent synergy here!

Thank you SAFS!

Enabling Rational XDE Tester

Enabling Rational XDE Tester for RRAFS

"Step right up, folks. Let me tell ya what we're gonna do:"

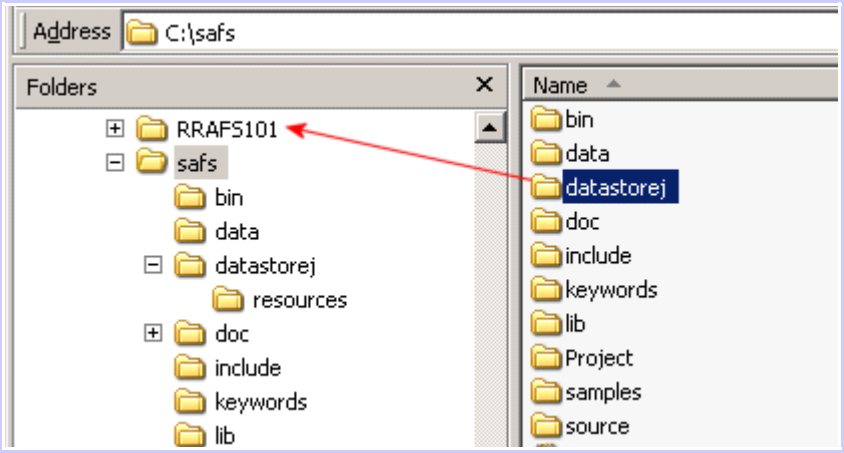
Steps 1 - 4 need to be done in order -- at least the first time.
These will detail how to prepare XDE Tester as a SAFS engine available to Rational Robot. This includes preparing a XDE Tester Datastore, the System CLASSPATH, the RRAFS.INI file, and running the XDE Tester Test Enabler.

Step 5 on configuring an XDE Tester Datastore for SAFS Development is not necessary unless you actually plan to debug, edit, and rebuild the SAFS Framework Java classes.

Create\Enable an XDE Tester Datastore for RRAFS Execution

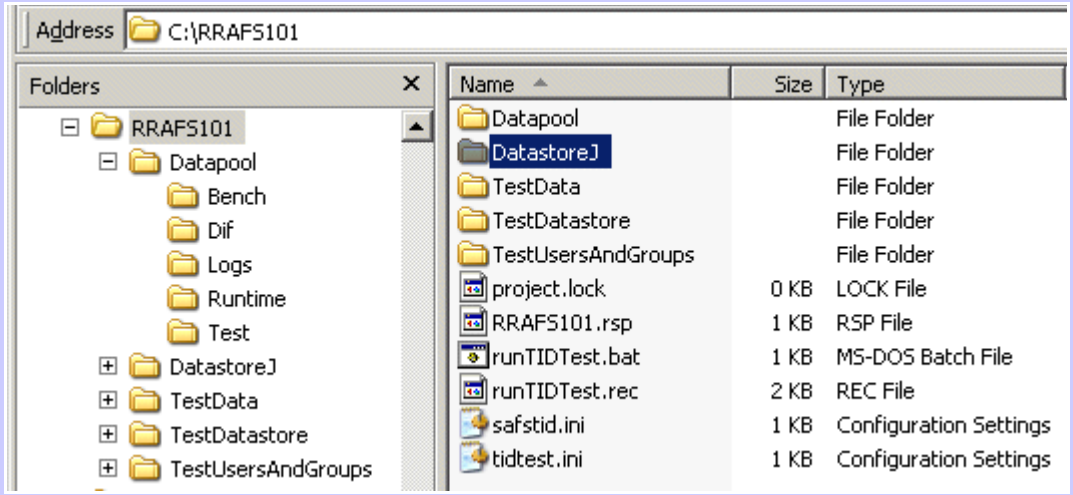
If only everything were as easy as this...

For this exercise we are assuming a XDE Tester Datastore that Robot can use to AUTOLAUNCH XDE Tester -- the SAFS/RobotJ engine. This is the minimum amount of work necessary to make the XDE Tester engine run.



Locate the SAFS**DatastoreJ** subdirectory where SAFS was installed. Copy the entire DatastoreJ folder into into the Robot projects root folder.
Using our RRAFS101 project as an example:

1. **Select** the SAFS **DatastoreJ** folder and then press **CTRL+C** on the keyboard. This copies the full folder contents to the clipboard.
2. **Select** the **RRAFS101** folder and then press **CTRL+V** on the keyboard. This pastes the stored DatastoreJ folder into our RRAFS101 folder.

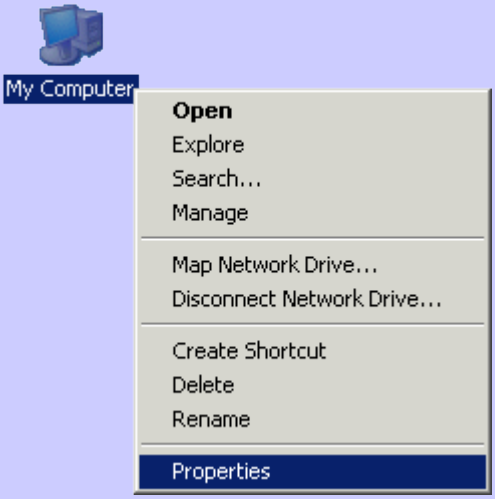


Review the RRAFS101 DatastoreJ Directory

Copying our entire DatastoreJ folder into the Rational project is the minimum required to make a functional XDE Tester Datastore. Of course, there is still some other setup to perform to make sure Rational Robot knows how to invoke XDE Tester. We'll be moving on to that next.

Finally, later on in the last section of this topic, "*Enabling Rational XDE Tester*", we will explore the Eclipse/Workbench setup necessary to make this Datastore operational for SAFS Development. These steps are not necessary right now, though. For now we are just concerned with execution, not development.

Review\Edit the System CLASSPATH

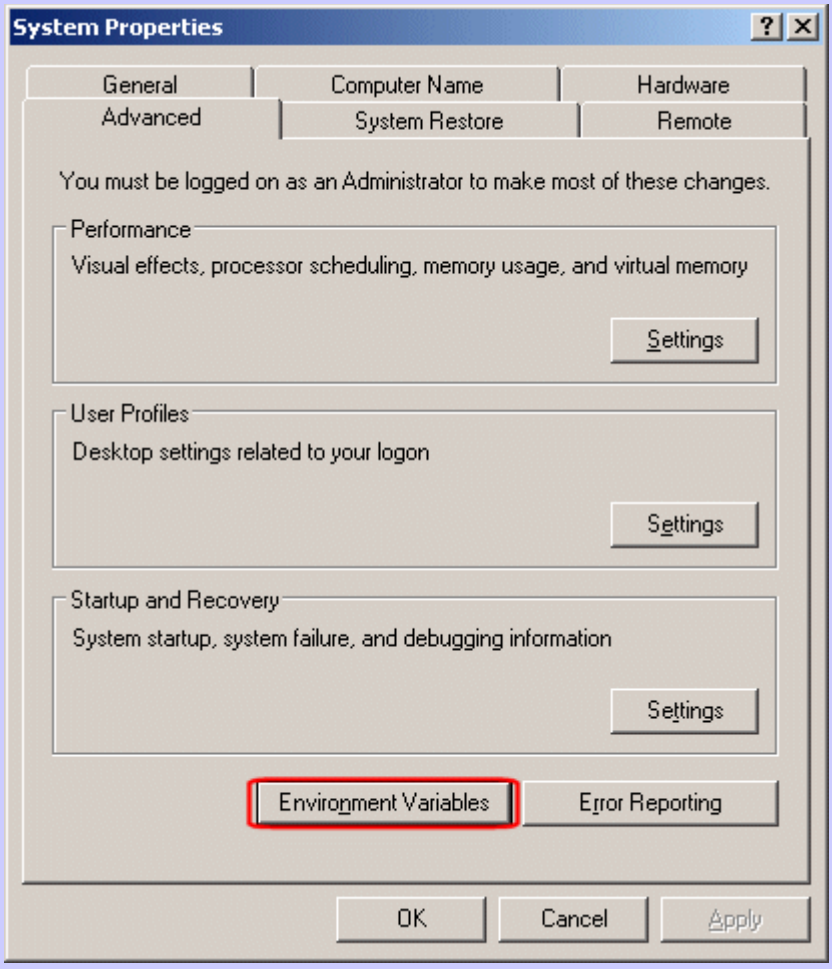


[Note: All the CLASSPATH settings discussed in this section are normally set by the RRAFS install scripts. If you know you had a fully successful install of RRAFS and SAFS then you don't need to review these settings.]

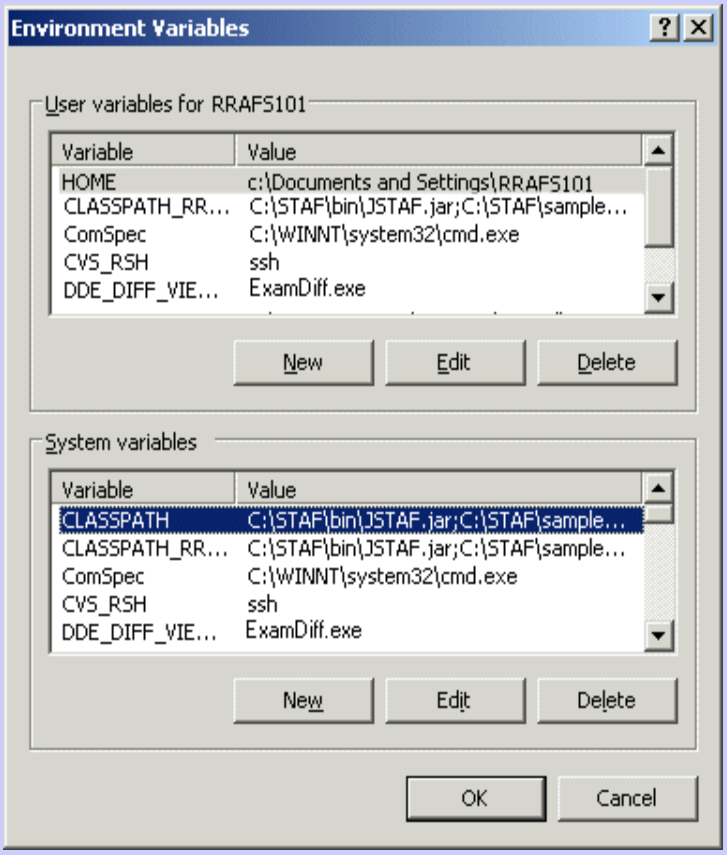
The System **Environment Variables** can be found in the System Properties Window. This is generally easiest to access on your system with a Right-Click on the "**My Computer**" icon and selecting "**Properties**".

Click on the **Environment Variables** button and proceed to the next page to review and/or edit the System **CLASSPATH**.

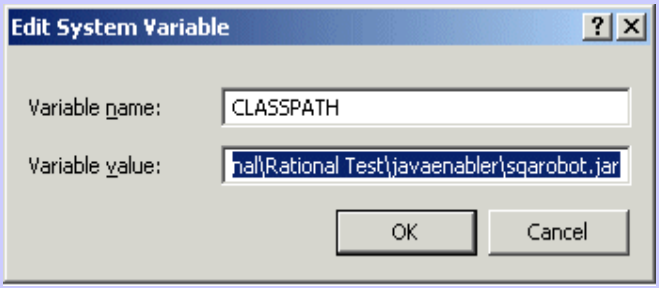
Hey! Not on the pictures here. On the icons and dialogs on YOUR PC!



Review or Edit the System CLASSPATH Setting



Having launched the Environment Variables dialog we can now review or Edit the **System CLASSPATH**. Note that most if not all of the settings we mention below should have already been made when we installed the framework. So there may actually be nothing to do but look and exit!



We *carefully* scroll the CLASSPATH Variable value editor and check that the JAR files listed below are present in the CLASSPATH. If they are not, we need to add the full path to each JAR file that is missing. It is also a good idea to have Rational Robot's Jave Enabler 'sqarobot.jar' in the CLASSPATH, too. You can find that one, I'm sure.

STAF\bin\JSTAF.JAR	**SAFS\lib\SAFS.JAR**
SAFS\lib\SAFSCUST.JAR	**SAFS\lib\jakarta-regexp-1.3.jar**
SAFS\lib\SAFSRATIONAL.JAR	***<rational_installdir>*\rational_ft.jar**
SAFS\lib\SAFSJREX.JAR	***<rational_installdir>*\xerces.jar**

<rational_installdir> *should contain the full path to the Rational* ***"com.rational.test.ft.wswplugin*"*** *directory that contains the specified Rational jar files.*

Exit the Environment Variables Dialog

Once you are satisfied with all the CLASSPATH settings you should SAVE or CANCEL and Close the Environment Variables editor. (CANCEL out if you did not make any changes.)

Now that wasn't so bad...was it?

Enabling XDE Tester in the RRAFS.INI

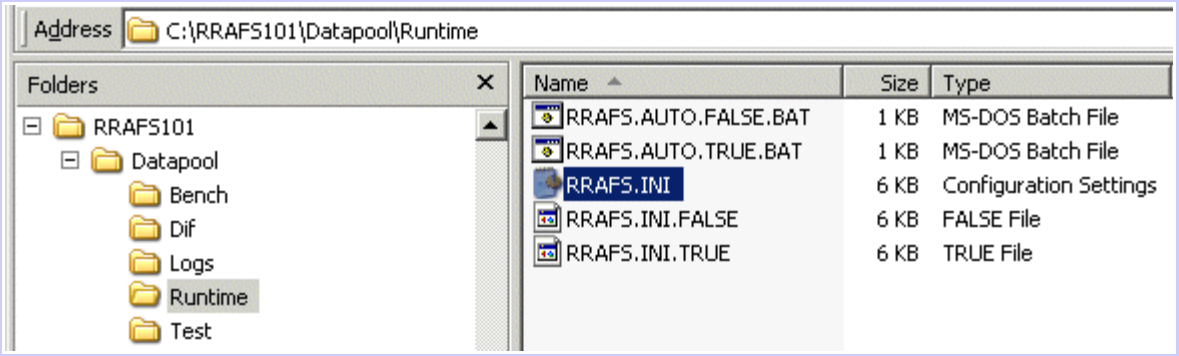
In a previous RRAFS101 lesson on "Enabling SAFS for RRAFS" we enabled just about every SAFS thing *except* XDE Tester. Well it is about high time we go that extra step. Note that those other SAFS services MUST be enabled for the SAFS/RobotJ engine to function properly. So if these services are NOT enabled then you must do that prior to (or while) proceeding. (Hint: All AUTOLAUNCH settings set to TRUE. Really difficult, we know.)

In your Robot project open the **Datapool\Runtime\RRAFS.INI** file. Review that the various SAFS services have their AUTOLAUNCH settings set to TRUE. For RRAFS.INI the sections in question are:

- SAFSVARS
- SAFSMAPS
- SAFSLOGS

In the next few pages we are going to edit the SAFS_ROBOTJ section of this RRAFS.INI to enable XDE Tester. You can view an online version of the RRAFS.INI template here.

For even more detail on some of these settings refer to the Command-Line Interface options in the Online Help for XDE Tester in the Help Menu.



Sample SAFS_ROBOTJ Section of RRAFS.INI:

```
      [SAFS_ROBOTJ]
      AUTOLAUNCH=TRUE
      INSTALLDIR="C:\Program
Files\Rational\XDETester\eclipse\plugins\com.rational.test.ft.wswplugin_2.0.0"
      DATASTORE=DatastoreJ
      PROJECT=DDEngine.rsp
      BUILD="Build 1"
      HOOKSCRIPT=TestScript
      LOGFOLDER=Default
      LOG=TestScript
      USERID=canagl
      ;PASSWORD=
```

Edit the SAFS_ROBOTJ setting for INSTALLDIR

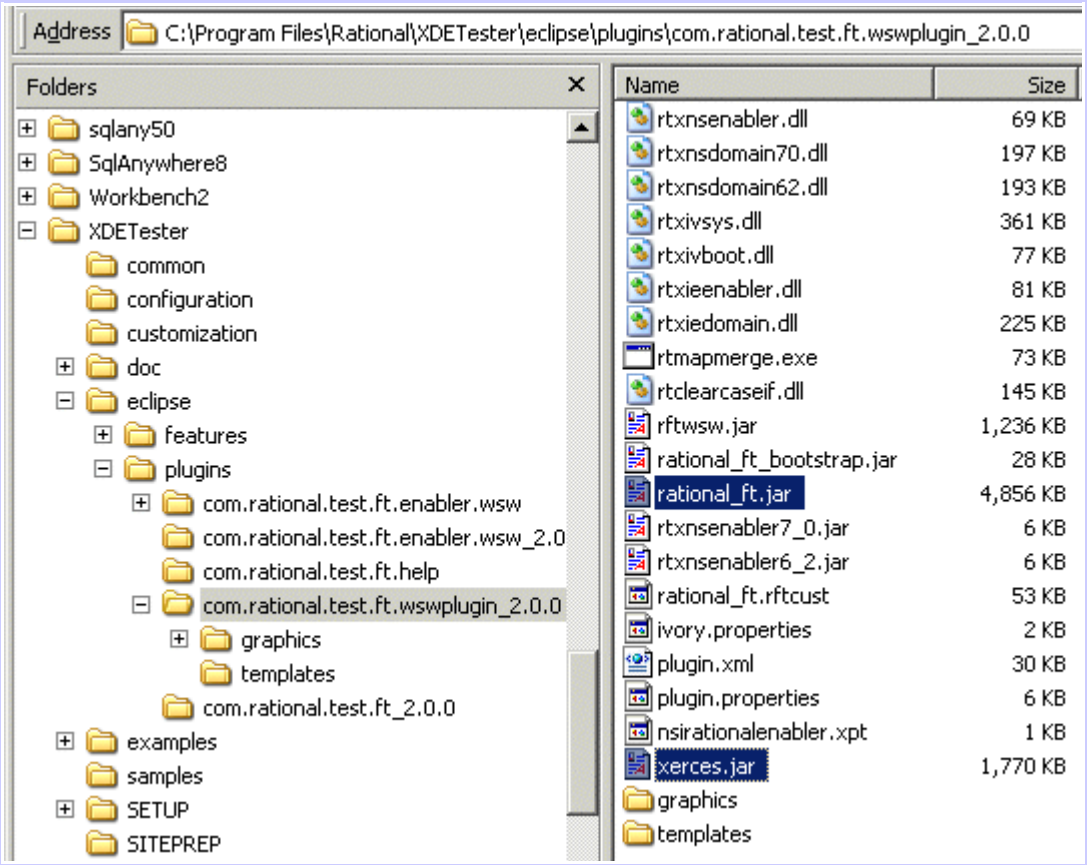
JVM and CLASSPATH settings will remain commented out. So lets start with **INSTALLDIR**.

Things will be fairly straightforward once we get past this one. The problem with **INSTALLDIR** is that Rational has been moving things around and it is hard to just say "you will find it here, or there". Depending on your install history of Rational products, and other factors we don't even know, you might be in for quite a file search.

INSTALLDIR needs to contain the full path to the "com.rational.test.ft.wswplugin" directory that contains the following Rational installed files:*

- *rational_ft.jar*
- *xerces.jar*

The RRAFS.INI template has 2 path entries to evaluate. One is pretty standard for RobotJ (V2002) while the other is a likely candidate for XDE Tester (V2003). But you will actually need to find the correct directory and enter the path appropriately. Don't forget to comment out or delete any inaccurate INSTALLDIR entries.

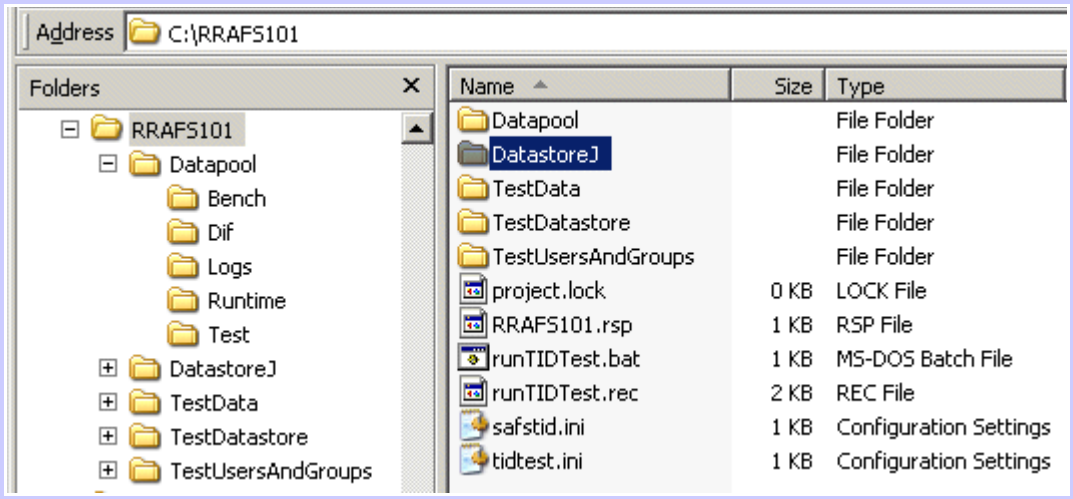


Please record below the full directory path where you found these jar files:

Edit the SAFS_ROBOTJ setting for DATASTORE

While every RRAFS.INI file will always point to the same INSTALLDIR, that is not the case for the **DATASTORE** entry. Each Robot project should have its own RRAFS.INI file, and each Robot project may have a different XDE Tester Datastore associated with it.

The **DATASTORE** setting must contain the directory that is your XDE Tester Datastore. We copied the SAFS DatastoreJ template at the beginning of this lesson in the section "Create\Enable an XDE Tester Datastore for RRAFS Execution". Remember?



If you have stuck with our recommended defaults throughout this tutorial, then you have a RRAFS101 Robot project with a "DatastoreJ" subdirectory acting as the XDE Tester Datastore.
And because that DatastoreJ directory is a subdirectory of our Robot project we can use a project-relative path like:

Datastore=DatastoreJ

If your XDE Datastore is NOT embedded within your Robot project then you will need to provide the full path to the XDE Tester Datastore for this Datastore setting.

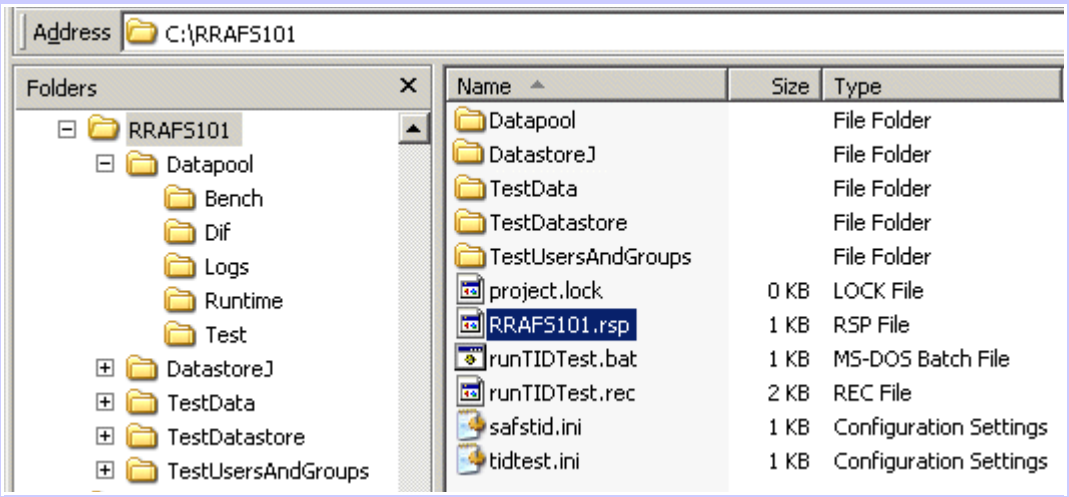
Please enter below the setting you entered for Datastore:

Edit the SAFS_ROBOTJ setting for PROJECT

The **PROJECT** setting is most certainly unique to each Robot project. Well, at least Rational Software believes it is going to be.

These needs to be the filename of the Robot project .RSP file -- the name of the Rational project with a ".rsp" extension. If somehow this is not in the root directory of the Rational project, then you will need to provide the full path to this file. In most cases the **PROJECT** setting will be something like:

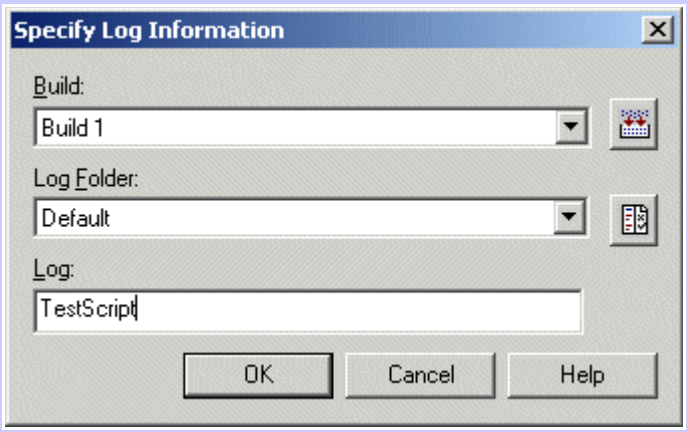
PROJECT=RRAFS101.rsp



Please enter below the filename\value you have provided for PROJECT:

Edit the SAFS_ROBOTJ setting for BUILD, LOGFOLDER, and LOG

The **BUILD**, **LOGFOLDER**, and **LOG** settings are specific to your Robot \ XDE Tester logging setup. These will contain the same values you would normally specify when prompted for Log information as you attempt to execute a Robot script:



In the case of our RRAFS101 project, we would not have ever changed the **BUILD** and **LOGFOLDER** from their default settings. But **LOG** is another matter. This can be anything.

For LOG we are going to specify the name of the XDE Tester script -- the "hook" -- we use to insert XDE Tester into the SAFS Framework. The name of that script is, ironically, "TestScript".

So, default values for these three settings would be:

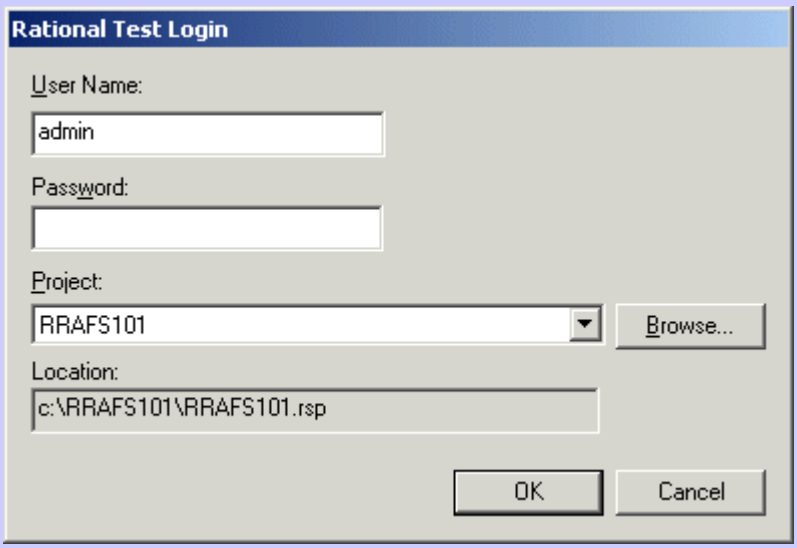
BUILD="Build 1"
LOGFOLDER=Default
LOG=TestScript

Now lets finish up with the last few settings on the next page.

Edit the SAFS_ROBOTJ settings for HOOKSCRIPT, USERID, and PASSWORD

The **HOOKSCRIPT** is the XDE Tester script we use to insert XDE Tester into the SAFS Framework. This predefined script is named "**TestScript**" and you can even see it in the XDE Tester Datastore root directory -- DatastoreJ, by default.

USERID and **PASSWORD** are those credentials you normally provide when you connect to the project.



If you don't normally supply a password, then leave this blank or commented out. So these 3 settings would be something like:

HOOKSCRIPT=TestScript
USERID=admin
PASSWORD=
OPTIONS=

Finally, the **OPTIONS** setting is only used by advanced testers familiar with the command-line interface of XDE Tester and know what needs to go here. So just leave this blank or commented out.

Review your SAFS_ROBOTJ Settings

We finish this section by reviewing all the settings. We must not forget that AUTOLAUNCH must be set to TRUE if we want Rational Robot to be able to handle launching XDE Tester automatically for us.

Assuming Rational V2003 and RRAFS101 default project settings, our SAFS_ROBOTJ section in RRAFS.INI should look something like this:

```
[SAFS_ROBOTJ]
AUTOLAUNCH=TRUE
;JVM=<path to java.exe>
;CLASSPATH=<alternate classpath for use with Java -cp argument>

INSTALLDIR="C:\Program
Files\Rational\XDETester\eclipse\plugins\com.rational.test.ft.wswplugin_2.0.0"

DATASTORE=DatastoreJ
PROJECT=RRAFS101.rsp
BUILD="Build 1"
HOOKSCRIPT=TestScript
LOGFOLDER=Default
LOG=TestScript
USERID=admin
PASSWORD=
OPTIONS=
```

Now let's move on to the next section (or take a well-deserved break!)

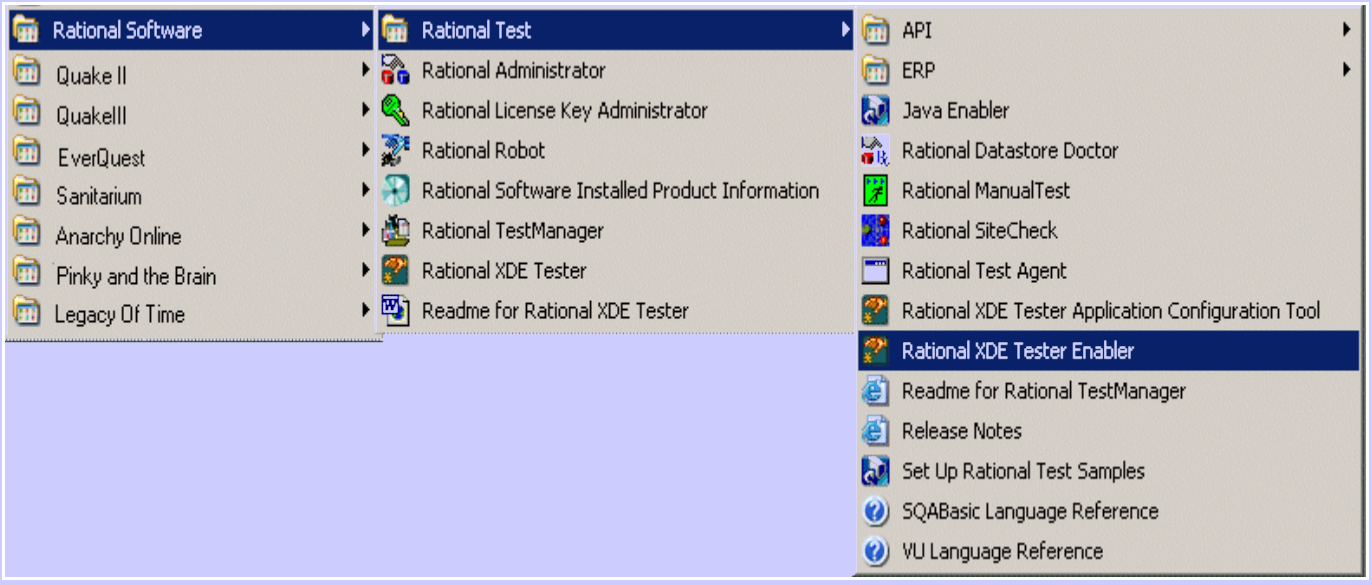
Launch the XDE Tester Test Enabler

Just as Rational Robot needs to inject itself into the application processes to snoop around so does XDE Tester. We must run the XDE Tester Test Enabler if we want it to 'click' or otherwise play with Java and Web components in our applications.

There are no less than 3 ways this Test Enabler can be run:

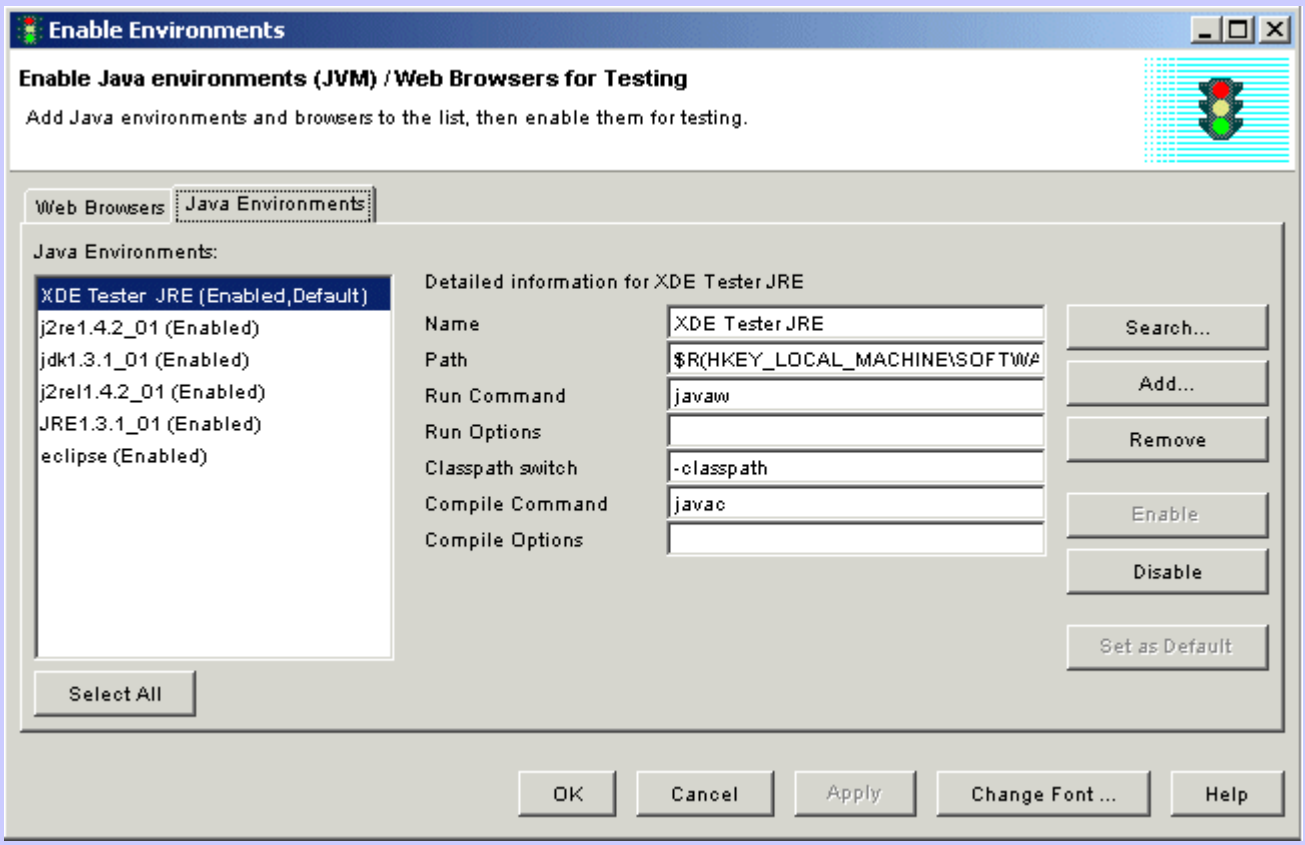
1. From the Rational Test Start Menu
2. From the XDE Tester Eclipse IDE
3. From the command-line

Since we are going for simplicity, we are just going to look at the first one:



XDE Test Enabler Application

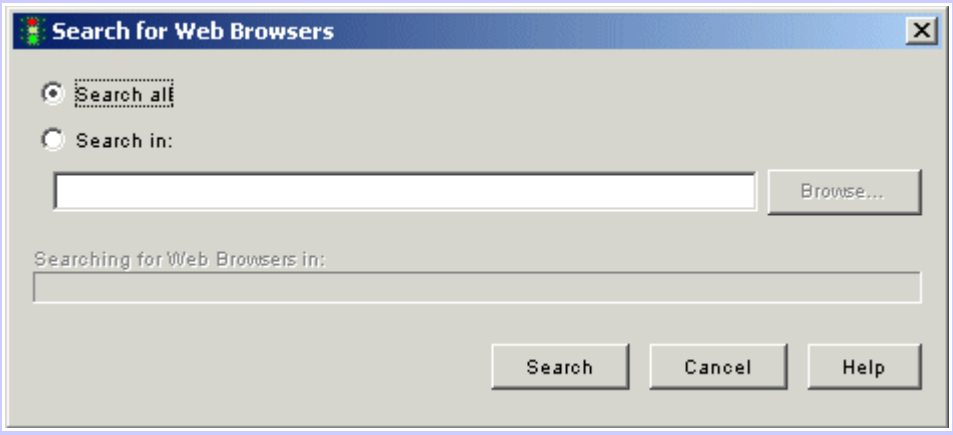
Once the Test Enabler application is running we can separately enable supported **Web Browsers** and **Java Environments**.



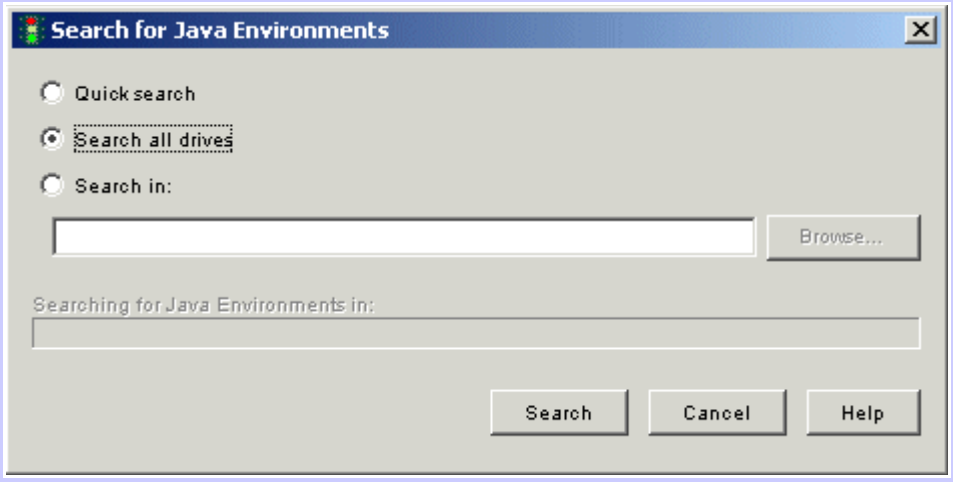
Enable Web Browsers and Java Environments

In turn select each Tab (**Web Browsers** and **Java Environments**) and the press the associated **Search...** button.

For Web Browsers we recommend you select "**Search all**", press the **Search** button, and then enable all browsers. Of course, you can customize this as necessary.



For Java Environments we recommend you select "**Search all drives**", press the **Search** button, and then enable all found JVMs. You can customize this, too.



Close the Test Enabler

Viola! That wasn't so hard, eh?

Once you have enabled everything you intend to enable you can close the Test Enabler application and proceed on to the last section: "*Create\Enable a XDE Tester Datastore for SAFS Development*".

Quiz: Enabling Rational XDE Tester

One of the more ambitious features available to RRAFS and any compatible Driver. How much of this did you retain? Let's find out.

1 1 Marks
What is the quickest method to prepare an existing Robot Project repository with the XDE Tester Datastore needed by RRAFS?

Answer:
- ○ a. Copy the SAFS\datastorej directory to the Robot Project root
- ○ b. Create an XDE Tester Datastore via the Eclipse IDE
- ○ c. Run SAFS\bin\SetupRobotJDatastore.vbs script

2 1 Marks
Where is the overriding Project RRAFS.INI optionally located?

Answer:
- ○ a. The RRAFS project's Datapool directory
- ○ b. The Robot project's root directory
- ○ c. DDE_RUNTIME: .\Rational\Rational Test\sqabas32 directory
- ○ d. The RRAFS project's Datapool\Runtime directory

3 1 Marks
In order to use the SAFS/RobotJ Engine for XDE Tester we *MUST* enable and/or use the SAFSMAPS, SAFSVARS, and SAFSLOGS services (True or False)?

Answer: ○ True ○ False

4 1 Marks
Select all of the following items that can be AUTOLAUNCHed via the RRAFS.INI file:

Answer:
- ☐ a. SAFSVARS
- ☐ b. SAFSLOGS
- ☐ c. SAFSMAPS
- ☐ d. SAFS/RobotJ
- ☐ e. SAFS/DriverCommands

5 1 Marks
What is the name of the special XDE Tester script that runs the SAFS/RobotJ Engine? This name must be set in the RRAFS.INI file as the [SAFS_ROBOTJ] 'HookScript' and is also the default value for the [SAFS_ROBOTJ] 'Log'.

Answer:

6 1 Marks

Besides setting AUTOLAUNCH=TRUE, what 3 very important items must be correctly edited to provide working values for launching the SAFS/RobotJ Engine?

Answer:
- ☐ a. OPTIONS
- ☐ b. INSTALLDIR
- ☐ c. CLASSPATH
- ☐ d. DATASTORE
- ☐ e. PROJECT

7 1 Marks

The XDE Tester Enabler normally used to enable Java JVMs and Web Browsers is NOT required to use the SAFS/RobotJ Engine (True or False)?

Answer: ○ True ○ False

8 1 Marks

In order to launch the SAFS/RobotJ Engine we must use the Eclipse/XDE Tester IDE to run the appropriate script (True or False)?

Answer: ○ True ○ False

Lesson

9

Enhancing, Extending, and Customizing

RRAFS has many features that allow the Automator to directly insert custom extensions and enhancements. In fact, certain features can be replaced in their entirety by such customizations. We'll provide a brief overview of these features and how they work.

How Custom Extensions Work

Quiz: Extending, Enhancing, and Customizing RRAFS

How Custom Extensions Work

How Customization Works

As good as RRAFS is, it has not yet implemented every thing that every one could possibly ever want to do. Because of this, the framework provides several libraries or "hooks" that facilitate adding extensions and custom features without interfering with the ability to accept future RRAFS upgrades. The user can add customizations without concern of a complicated "merge" process with a future version of RRAFS.

Within RRAFS the following libraries are provided to facilitate different types of customization:

- **CustomRecordTypes**
- **CustomDriverCommands**
- **CustomTestCommands**
- **CustomLogUtilities**
- **CustomStatusUtilities**

In addition to these, RRAFS provides a general-purpose *CustomUtilities.SBL* library with associated *CustomUtilities.SBH* header. These don't do anything by themselves. However, The CustomUtilities.SBH header is automatically included in the *DDEngine.SBH* header provided by RRAFS.

'$Include: "DDEngine.SBH"

Any script or library that includes DDEngine.SBH gains access to most every useful feature provided by the RRAFS framework *AND* any custom routines Declared in CustomUtilities.SBH.

Protecting Customizations during RRAFS Upgrades

With each RRAFS release a clean do-nothing copy of all the various Custom Extensions is redistributed. There are two ways to protect any customizations we might have made in these libraries. This is important because if we have made customizations RRAFS will overwrite them if we don't take the necessary precautions.

The first method of protection is to copy any modified Custom Extensions to the Robot repository's TMS_Scripts\SQABas32 directory. Files there override files in the DDE-RUNTIME directory (.\Rational\Rational Test\sqabas32) and RRAFS does not copy any files into any Robot repository.

The second method of protection is to make a temporary backup of all Custom Extensions you have modified before upgrading to a new release of RRAFS. Once the upgrade is complete you can restore the modified Custom Extensions into the DDE_RUNTIME directory overwriting the clean do-nothing versions from the new release.

These steps are only necessary if you have actually implemented customizations in the Custom Extension libraries. If you have not, then you need do nothing.

So let's start out learning about the most flexible wide-open custom hook of them all:
CustomRecordTypes...

CustomRecordTypes Overview

Follow the link to review the documentation for CustomRecordTypes. It isn't much but it does tell you the libraries involved and some info on how and when Function *CustomDDERecord* in CustomRecordTypes is invoked.

What isn't readily obvious is the wide-open do-anything-you-want capability this offers. Note the following quote from the CustomRecordTypes doc concerning matching the record type -- field #1:

> *"If no match is found, the DDE then tries to match the field to the name of a valid SCRIPT in the current project. If no matching script is found, the DDE routes the record to this library to see if the user has implemented a matching custom record type."*

This means the tester is free to implement any new type of record or test input in any format or syntax they find most appropriate. Upon encountering this unknown data RRAFS will pass it along to the CustomRecordTypes library for processing. Of course, the default implementation provided with RRAFS does nothing and simply returns "DDU_SCRIPT_NOT_EXECUTED" -- which is the normal "didn't do anything with this" response.

Let's see a working example that is actually coded into the CustomRecordType library...

Custom Record Types
http://safsdev.sourceforge.net/sqabasic2000/CustomRecordTypes.htm

CustomDDERecord Working Example

Go ahead and open the CustomRecordTypes.SBL file from your DDE_RUNTIME directory (.\Rational\Rational Test\sqabas32). There you will find the full do-nothing implementation that is ready to be enhanced. A small working example in the code provides a sample implementation and is copied below:

```
Case "CRT"

    Result = DDU_NO_SCRIPT_FAILURE

    'this sample record type happens to log a TEST Pass or Fail
    AUIncrementTestPasses statusInfo
    LogMessage "Sample Custom Record Type processing.", .fac, PASSED_MESSAGE
    .statuscode = Result
```

You can actually place a record whose record type (field #1) is "CRT" into a working test table and, assuming there is no script named "CRT", this CustomDDERecord code will be executed and a successful test will be logged and counted. Go ahead and try it!

Note: "CRT" just happens to be a working example in this library. This can be extended to provide real functionality or it can be removed or changed to ANYTHING.

"Please click on any button at all."

As shown above, the custom record need not even have fields, per se. The CustomDDERecord function will be called and it can be custom coded to do anything the tester desires.

Let's now move on to CustomDriverCommands.

CustomDriverCommands Overview

Follow the link to review the documentation for CustomDriverCommands. It isn't much but it does tell you the libraries involved and some info on how and when Function *CustomDDEDriverCommand* in CustomDriverCommands is invoked.

Note the following quote from the CustomDriverCommands doc:

> *The DDE will first try to match the driver command (field #2) to core DDE driver commands. If no match is found, the DDE routes the record to this library to see if the user has implemented a matching custom driver command.*

Thus, the tester can implement new Driver Commands to do just about anything they desire. While field #1 must be a valid Driver Command record type (C,CW,CF), the rest of the record can be anything in any format or syntax needed by the tester. Upon encountering this unknown data RRAFS will pass it along to the CustomDriverCommands library for processing. Of course, the default implementation provided with RRAFS does nothing and simply returns "DDU_SCRIPT_NOT_EXECUTED" -- which is the normal "didn't do anything with this" response.

Let's see a working example that is actually coded into the CustomDriverCommands library...

Custom Driver Commands
http://safsdev.sourceforge.net/sqabasic2000/CustomDriverCommands.htm

CustomDDEDriverCommand Working Example


Go ahead and open the CustomDriverCommands.SBL file from your DDE_RUNTIME directory (.\Rational\Rational Test\sqabas32). There you will find the full do-nothing implementation that is ready to be enhanced. A small working example in the code provides a sample implementation and is copied below:

```
Case "CDC"

    Result = DDU_NO_SCRIPT_FAILURE
    LogMessage "Custom Driver Command processed.", .fac
    .statuscode = Result
```

C, CDC

You can actually place a Driver Command record with field #2 as "CDC" into a working test table and this CustomDDEDriverCommand code will be executed. The message will be logged and the record will be counted. Go ahead and try it!

Note: "CDC" just happens to be a working example in this library. This can be extended to provide real functionality or it can be removed or changed to ANYTHING.

"C, Do something amazing against our application!"

As shown above, the custom Driver Command need not even have fields, per se. The CustomDDEDriverCommand function will be called and it can be custom coded to do anything the tester desires.

Let's now move on to CustomTestCommands.

CustomTestCommands Overview

Follow the link to review the documentation for CustomTestCommands. The details of the libraries involved and detailed info on how and when Function *CustomDDETestCommand* in CustomTestCommands is invoked.

Note the following quote from the CustomTestCommands doc:

> *It is important to note that this library is NOT used to extend Cycle and Suite level test commands. This library only extends the ability to implement additional Component Functions for the Step Driver test level.*

Thus, the tester can implement new Component Functions to do just about anything they desire.

Field #1 must be a valid Step level test record type (T,TW,TF). Field #2 and Field #3 contain valid Window and Component values as would be typically defined in the App Map. It is Field #4 that contains the newly defined custom action that will ultimately be processed by the CustomTestCommands library. The rest of the record can be anything in any format or syntax needed by the tester.

Upon encountering this unknown command RRAFS will pass it along to the CustomTestCommands library for processing. Of course, the default implementation provided with RRAFS does nothing and simply returns "DDU_SCRIPT_NOT_EXECUTED" -- which is the normal "didn't do anything with this" response.

Let's see a working example that is actually coded into the CustomTestCommands library...

Custom Test Commands
http://safsdev.sourceforge.net/sqabasic2000/CustomTestCommands.htm

CustomDDETestCommand Working Example

Go ahead and open the CustomTestCommands.SBL file from your DDE_RUNTIME directory (.\Rational\Rational Test\sqabas32). There you will find the full do-nothing implementation that is ready to be enhanced. A small working example in the code provides a sample implementation and is copied below:

```
Case "CTC"

    Result = DDU_NO_SCRIPT_FAILURE
    LogMessage "Custom Test Command "& .testcommand &" processed.", .fac, PASSED_MESSAGE
    AUIncrementTestPasses statusInfo
    .statuscode = Result
```

T, AWindow, AComponent, CTC

As shown above, we can actually place a valid Step level test record with field #4 as "CTC" into a working test table and this CustomDDETestCommand code will be executed. The test success will be logged and counted. Go ahead and try it!

Note: "CTC" just happens to be a working example in this library. This can be extended to provide real functionality or it can be removed or changed to ANYTHING.

T, AWindow, AComponent, "Do some real fancy stuff here!"

As shown above, the custom Test Command need not have fields after field #3. The CustomDDETestCommand function will be called and it can be custom coded to do anything the tester desires.

Let's now move on to CustomLogUtilities.

CustomLogUtilities Overview

Follow the link to review the documentation for CustomLogUtilities. The details of the libraries involved and detailed info on how and when the provided Functions are invoked.

The functions are:

- *CustomDDELogInitialization*
- *CustomDDELogMessage*
- *CustomDDELogFinalization*

Note the following quote from the CustomLogUtilities doc:

> *This module is used to implement project specific, or site specific logging that is intended to either enhance or replace the built-in logging provided by the Core framework.*

Thus, the tester can implement logging enhancements to do just about anything they desire. We will now go into more detail for each of these routines...

http://safsdev.sourceforge.net/sqabasic2000/CustomLogUtilities.htm

CustomDDELogInitialization Overview

Follow the link to review the documentation for CustomDDELogInitialization. This provides a short overview of the function and the return codes used upon exiting.

Before exiting normal log initialization with InitLogFacility in LogUtilities, the CustomDDELogInitialization function is called to allow a tester to add customizations. This might include writing custom information to the log, initializing additional site-specific logs, or any other customization needed by the tester. The return code from CustomDDELogInitialization can even instruct normal logging to be bypassed entirely (return code -1).

By default, with no customizations in place, CustomDDELogInitialization simply returns with the equivalent of "continue logging normally" (return code 0).

Go ahead and open the *CustomLogUtilities.SBL* file from your DDE_RUNTIME directory (.\Rational\Rational Test\sqabas32). There you will find the sparse do-nothing implementation of the CustomDDELogInitialization function that is ready to be enhanced:

```
CustomDDELogInitialization = 0
```

If you are so inclined to experiment, you can add a line to the CustomDDELogInitialization routine to actually print out a message upon initialization:

```
LogMessage "Custom Initialization Complete!", fac
CustomDDELogInitialization = 0
```

Remember that you must compile any changes to the CustomLogUtilities library before they will take affect. You will see this message appear near the top of each log initialized through LogUtilities in any subsequent test.

CustomDDELogInitialization
http://safsdev.sourceforge.net/sqabasic2000/CustomLogUtilities.htm#customddeloginitialization

InitLogFacility in LogUtilities
http://safsdev.sourceforge.net/sqabasic2000/LogUtilities.htm#initlogfacility

CustomDDELogMessage Overview

Follow the link to review the documentation for CustomDDELogMessage. This provides a short overview of the function and the return codes used upon exiting.

Before logging any message with LogMessage in LogUtilities, the CustomDDELogMessage function is called to allow a tester to add customizations. This might include writing custom information to a file, the console, or custom output; adding additional information to the standard logs, or any other customization needed by the tester. The return code from CustomDDELogMessage can even instruct normal logging to be bypassed entirely (return code -1).

By default, with no customizations in place, CustomDDELogMessage simply returns with the equivalent of "continue logging normally" (return code 0).

Go ahead and open the *CustomLogUtilities.SBL* file from your DDE_RUNTIME directory (.\Rational\Rational Test\sqabas32). There you will find the sparse do-nothing implementation of the CustomDDELogMessage function that is ready to be enhanced:

```
CustomDDELogMessage = LU_NORMAL_DDE_LOGGING
```

If you are so inclined to experiment, you can add a line to the CustomDDELogMessage routine to actually print out a message to the SQAConsole:

```
SQAConsoleWrite "Custom Logging Complete!"
CustomDDELogMessage = LU_NORMAL_DDE_LOGGING
```

Remember that you must compile any changes to the CustomLogUtilities library before they will take affect. You will see this message appear in the console for *every* message logged through LogUtilities in any subsequent test.

CustomDDELogMessage
http://safsdev.sourceforge.net/sqabasic2000/CustomLogUtilities.htm#customddelogmessage

LogMessage in LogUtilities
http://safsdev.sourceforge.net/sqabasic2000/LogUtilities.htm#logmessage

CustomDDELogFinalization Overview

Follow the link to review the documentation for CustomDDELogFinalization. This provides a short overview of the function and the return codes used upon exiting.

Before closing logs from a Robot script with CloseAllLogs in LogUtilities, the CustomDDELogFinalization function is called to allow a tester to add customizations. This might include writing final custom information to a file, the console, or custom output; adding additional information to the standard logs, or any other customization needed by the tester. The return code from CustomDDELogFinalization can even instruct the normal log shutdown mechanism to be bypassed entirely (return code -1).

By default, with no customizations in place, CustomDDELogFinalization simply returns with the equivalent of "continue shutdown normally" (return code 0).

Go ahead and open the *CustomLogUtilities.SBL* file from your DDE_RUNTIME directory (.\Rational\Rational Test\sqabas32). There you will find the sparse do-nothing implementation of the CustomDDELogFinalization function that is ready to be enhanced:

```
CustomDDELogFinalization = LU_NORMAL_DDE_LOGGING
```

If you are so inclined to experiment, you can add a line to the CustomDDELogFinalization routine to actually print out a message to the closing log:

```
LogMessage "Custom Logging Complete!"
CustomDDELogFinalization = LU_NORMAL_DDE_LOGGING
```

Remember that you must compile any changes to the CustomLogUtilities library before they will take affect. You will see this message appear just prior to log shutdown whenever CloseAllLogs is invoked in the Robot Script that launches the test.

Let's finish up this lesson by discussing CustomStatusUtilities...

CustomDDELogFinalization
http://safsdev.sourceforge.net/sqabasic2000/CustomLogUtilities.htm#customddelogfinalization

CloseAllLogs in LogUtilities
http://safsdev.sourceforge.net/sqabasic2000/LogUtilities.htm#closealllogs

CustomDDEStatusCounter Overview

Follow the link to review the documentation for CustomDDEStatusCounter. This provides a short overview of the subroutine used to perform custom status tracking. Contrary to the library's header block, the routine does not allow us to disable or bypass normal status tracking. It simply allows us to add custom status tracking.

After incrementing all standard and user-defined status counters with AUIncrementStatusCounters in ApplicationUtilities, the CustomDDEStatusCounter subroutine is called to allow a tester to add customizations. This might include writing custom information to a separate file, the console, or elsewhere; adding additional information to the standard logs, or any other customization needed by the tester.

By default, with no customizations in place, CustomDDEStatusCounter does absolutely nothing. Since it is a subroutine it does not return any value.

Go ahead and open the *CustomStatusUtilities.SBL* file from your DDE_RUNTIME directory (.\Rational\Rational Test\sqabas32). There you will find the empty implementation of the CustomDDEStatusCounter routine that is ready to be enhanced.

The routine has access to the current AUStatusInfo structure and the status value of the in-process record.

If you are so inclined to experiment, you can add a line to the CustomDDEStatusCount routine to actually print out a message to the SQAConsole:

SQAConsoleWrite "Custom Status Info recorded!"

Remember that you must compile any changes to the CustomStatusUtilities library before they will take affect. You will see this message appear in the console for *every* status count incremented through AUIncrementStatusCounters in ApplicationUtilities. Normally that is at least once for every record in the test.

CustomDDEStatusCounter
http://safsdev.sourceforge.net/sqabasic2000/CustomStatusUtilities.htm#customddestatuscounter

AUIncrementStatusCounters in ApplicationUtilities
http://safsdev.sourceforge.net/sqabasic2000/ApplicationUtilities.htm#auincrementstatuscounters

AUStatusInfo Structure
http://safsdev.sourceforge.net/sqabasic2000/ApplicationUtilities.htm#user_defined

In-process Status Value
http://safsdev.sourceforge.net/sqabasic2000/ApplicationUtilities.htm#constants

Quiz: Extending, Enhancing, and Customizing RRAFS

There are many ways to add your own features to RRAFS. Do you remember them all? Let's see how much you actually do remember!

1 1 Marks
Select the one SQABasic HEADER file that provides your scripts and libraries access to virtually everything the RRAFS Framework has to offer:

Answer:
- ○ a. SQAUtil.SBH
- ○ b. WIN32.SBH
- ○ c. StepDriver.SBH
- ○ d. DDEngine.SBH

2 1 Marks
Backups of Custom Extensions are *not* necessary when performing a RRAFS upgrade (True or False)?

Answer: ○ True ○ False

3 1 Marks
What extension library allows us to provide entirely custom record formats and syntax?

Answer:
- ○ a. CustomDriverCommands
- ○ b. CustomTestCommands
- ○ c. CustomRecordTypes

4 1 Marks
Which extension library allows us to provide site-unique Driver Commands available to all test levels?

Answer:
- ○ a. CustomRecordTypes
- ○ b. CustomTestCommands
- ○ c. CustomDriverCommands

5 1 Marks
Which extension library allows us to add site-unique Component Functions -- available only at the Step Level?

Answer:
- ○ a. CustomDriverCommands
- ○ b. CustomRecordTypes
- ○ c. CustomTestCommands

6 1 Marks
Which extension library allows us to enhance or even replace the logging provided by RRAFS?

Answer:
- ○ a. CustomLogUtilities
- ○ b. CustomUtilities
- ○ c. CustomStatusUtilities

7 1 Marks

Which extension library allows us to capture running status information and provide custom responses or activities relating to it?

Answer:

- a. CustomCounterCommands
- b. CustomStatusUtilities
- c. CustomLogUtilities
- d. CustomCounterUtilities

Lesson

10 Future Features and Fancies

The SAFSDEV group is neve done adding new features and functions. In fact, one of their goals is to expand upon the wide array of testing options available through SAFS. Step inside and see what is in store for RRAFS and the associated SAFS Framework.

RRAFS: The Ongoing Evolution

RRAFS: The Ongoing Evolution

RRAFS: The Big TOE

So, you might be asking yourself, "What is this? The Big TOE?"

Well, "RRAFS: The Big The Ongoing Evolution", that just wasn't going to cut it! How about, "RRAFS: TOE"? Perhaps that is better. So let's talk about TOE.

The evolution of RRAFS has been in high gear since Day 1. Originally entirely coded in Robot SQABasic, RRAFS has endured several evolutionary leaps over the years.

The first mutation removed large chunks of RRAFS functionality out of SQABasic code and into ActiveX/COM objects provided by the DDVariableStore.DLL. This DLL was the first attempt to centralize large amounts of functionality into reusable COM objects for other tools -- not just Rational Robot. There was a big party with lots of fanfare -- but nobody came.

The DLL worked well, yet the nature of DLLs and COM meant we were limiting ourselves to COM and the Windows Operating System. And the soothsayers suggested we lacked real vision. Actually, we lacked time and resources.

We began to hear cries for more than this. Automators wanted to use what was then called Rational RobotJ -- and Java. These were areas that COM and the Windows OS feared to tread. Another evolutionary leap was needed and our investigation and research culminated in the design proposal of the STAF-based SAFS Framework. More importantly, some of these automators actually stepped forward and said, "We'll help you build it." So we did!

Features provided by the DDVariableStore.DLL could now be provided by SAFS. And the SAFS features were multi-lingual and multi-platform. From within RRAFS we could now talk to external tools based on Java, VB, C/C++, Perl, Python and others. We now had a platform from which we could jump to Rational RobotJ and a host of other options, and a host of other hosts. And it was good.

Today the evolution continues...

STAF
http://staf.sourceforge.net

RRAFS: TOE Continues

With the SAFS Framework firmly in place, RRAFS grew elegant fingers that could reach into odd places. It could now tap IBM Rational XDE Tester and we could use both tools in the course of testing. New Driver Commands were developed entirely in Java and made available in yet another tool called SAFS/DriverCommands. RRAFS grew to strum that tune, too!

RRAFS has long been the king of the SAFSDEV dynasty. However, using the new SAFS Framework tools -- including IBM Rational XDE Tester and SAFS/DriverCommands -- was constrained by this dependency on RRAFS. And IBM Rational Software was no longer providing both IBM Rational Robot and IBM Rational XDE Tester in a bundled package like it did in the days before The Great Renaming of Names.

Some automators needed to use these separate SAFS gems without RRAFS, but there was no other tool capable of acting as the overall controller or Driver. This led to the construction of SAFSDRIVER -- a Java-based Driver duplicating the controller/driver functions provided by RRAFS.

SAFSDRIVER is relatively new, and it is not yet a 100% capable replacement for all Driver functions provided by RRAFS, but it soon should be. So what does this mean for the TOE of RRAFS? Is it becoming extinct?

Absolutely not! The future of TOE is straight ahead...

RRAFS: TOE and You

RRAFS will always be Driver-capable for automators that want to stay inside the RRAFS box. But it is the Engine side of RRAFS that provides the great GUI automation features that are most prized, and RRAFS will continue to grow and evolve as a powerful full-featured Engine.

SAFSDEV hopes to eventually enhance RRAFS so that it can be externally controlled by SAFSDRIVER in place of its own built-in Driver. This will enable RRAFS developers to focus more on robust GUI automation features and less on Driver details -- such as implementing RRAFS-specific support for each new SAFS Engine as they become available. (It is better to implement new Engine support in one multi-platform SAFSDRIVER instead of duplicating the effort in both SAFSDRIVER *and* RRAFS.)

The ultimate goal for RRAFS is to become a standard SAFS Engine controlled by any external SAFS-compatible Driver like SAFSDRIVER. This is exactly how the Engine for IBM Rational XDE Tester is implemented. In fact, the XDE Tester Engine has no Driver capabilities at all. It *must* be controlled by an external Driver like SAFSDRIVER or RRAFS.

Beyond these lofty goals, the evolution of RRAFS will continue along whatever lines the active tester/developer community dictates. New features and functions are contributed on a regular basis. More complete .NET support is trickling in as .NET testers identify and fix .NET issues. The same is true for support of Java and Web client testing.

Stay tuned to the SAFSDEV-RRAFS forum to keep abreast of what lies in store. And don't forget: You can help contribute to the TOE of RRAFS, too!

SAFSDEV-RRAFS Forum
http://lists.sourceforge.net/lists/listinfo/safsdev-rrafs