Getting Started With STAF

Getting Started With STAF

Version 3.0.3

Document Owner: David Bender

1. Introduction

1.1. Overview

STAF is an Open Source automation framework designed around the idea of reusable components. It is intended to make it easier to create automated testcases and workloads. STAF can help you increase the efficiency, productivity, and quality of your testing by improving your level of automation and reuse in your individual testcases as well as your overall test environment.

This document will guide you through many common tasks that are performed when using STAF, including a detailed examination of a Demo which shows how you can instrument and leverage STAF in your testcases.

Note that this document is based on STAF V3.0.0. Older releases of STAF may not have the same functionality that is described in this document.

1.2. STAF Resources

1.2.1. Website

Here is a link to the official STAF SourceForge website From this web site you can access and contribute to STAF software and documentation, as well as submit Bug and Feature requests. There are also Public Forums where you can ask questions about STAF

1.2.2. Forums

You can ask questions about STAF on the Help forum on the STAF website.

1.2.3. Mailing Lists

There are 3 Mailing Lists on the SourceForge STAF web site for which you can subscribe:

- staf-news Low traffic, read-only list for news and announcements
- staf-users Questions, suggestions, support, and general discussion of STAF
- staf-devel For development use only

1.2.4. IBM Internal newsgroup

The IBM Intranet STAF news group allows IBM Employees to participate in discussions regarding STAF and to obtain assistance both from the STAF Support team and from fellow STAF users. Many times when you have a question about STAF or have run into a STAF problem, you'll find that you're not the first user with that same question or problem, so you may be able to get assistance just by browsing through the existing discussions.

The news group name is ibm.websphere.awe and is located on news server websphere.austin.ibm.com.

1.3. Installing STAF

1.3.1. Windows Installation

We provide InstallShield Universal installers for Windows. You can perform a GUI installation, or specify the -silent option for a silent installation. See the STAF User's Guide Windows Installation section for more details.

1.3.2. Unix Installation

We provide InstallShield Universal installers for Unix platforms that are supported by InstallShield. You can perform a GUI installation, or specify the -silent option for a silent installation. See the STAF User's Guide Unix Installation section for more details.

For all Unix platforms, we provide a single compressed tar file, or gzipped tar file, that can be used to install STAF. Before installing STAF, you will need to uncompress (or gunzip) the compressed file and then untar it. See the STAF User's Guide Unix Installation section for more details.

1.3.3. Notes

Throughout this document we will assume that you installed STAF to the default location (C:\STAF on Windows, /usr/local/staf on Unix). If you installed STAF to another location, you will need to make the appropriate substitutions.

This document will show Windows path information by default (and screen captures will be from Windows systems). If you are using Unix you will need to make the appropriate path translations.

2. Basic STAF Concepts

2.1. STAFProc

STAF runs as a daemon process (called STAFProc) on each system. So, for example, if you wanted to run STAF on your office machine and 5 test machines in a lab, you would install STAF on all 6 systems. Then, to use STAF in this environment, you would start STAFProc on all 6 machines. The collection of machines on which you have installed STAF is referred to as the STAF Environment.

STAF operates in a peer-to-peer environment; in other words, there is no client-server hierarchy among machines running STAF. Figure 1 illustrates that the STAFProc daemons serve as the communication mechanism over the network.

Figure 1.

2.2. STAF Services

STAF *services* are reusable components that provide all the capability in STAF. Each STAF service provides a specific set of functionality (such as Logging, File Transfer, Process Inovcation, etc.) and defines a set of requests that it will accept.

2.3. STAF Service Requests

STAF Services are used by sending STAF *requests* to them. A STAF request is simply a string which describes the operation to perform. STAF requests can be sent to services on the local machine or to another, remote, machine in the STAF Environment. In either case, the STAFProc daemon process handles the sending and receiving of requests.

2.4. STAF Machine Names

Machine names are used to identify different systems in the STAF Environment. Typically, STAF machine names are simply the TCP/IP host name or the IP address of the machine.

2.5. STAF Instances

Since multiple instances of STAF can be run at the same time on the same system, a STAF *Instance name* is used to specify a name for each STAF instance. You specify the instance name to be used by setting the environment variable STAF_INSTANCE_NAME. The default instance name is "STAF".

2.6. STAF Handles

A *handle* is a unique identifier which is used when submitting requests to STAF. This handle, combined with the STAF instance name, uniquely identifies a particular process in the STAF environment.

It is this combination of STAF instance name and handle that allows STAF Services to track requests from multiple processes on different machines. Every process that accesses STAF does so through a handle.

2.7. STAF Variables

STAF provides facilities to store and retrieve *variables*. These variables are commonly used to store Testcase configuration information, Runtime information, and System Environment information.

These variables live within the STAFProc process. This allows them to be dynamically updated without having to start and stop applications using them (after the update, any applications referencing the updated variable will get the new value).

STAF maintains a "system" variable pool that is common to all the handles on a given STAF Client. STAF also maintains a "shared" variable pool which is also system-wide, but which will be sent across the network and used in variable resolution on remote systems. In addition, each handle has its own variable pool.

By default, the values of variables in a handle's variable pool override the values of variables in the system and shared variable pools. However, the handle may override this behavior when asking for the value of a variable. Basically, as part of every remote request, the originating handle and system shared variable pools are sent across the wire. These pools are

stored only for the duration of the request for use in variable resolution.

2.8. STAF Security

Security in STAF can be defined at the machine level and/or the user level. In other words, you grant access to machines and/or to userids.

Access in STAF is granted by specifying a certain trust level for a machine or user, where trust level 0 indicates no access and trust level 5 indicates full access.

Each service in STAF defines what trust level is required in order to use the various functions the service provides.

2.9. STAF Queues

Each handle in STAF has a priority *queue* associated with it.

Applications receive messages sent from other handles on their queue.

2.10. Submitting STAF Requests

While STAF requests can be submitted from a variety of programming languages, they may also be submitted from the command line (via the STAF executable, which is described in more detail later in this document).

However, submitting requests to STAF from the command line does have its limitations. When you submit a request to STAF from the command line, a unique handle is generated for that request. After the request completes, that handle is no longer active in STAF. So if you were to submit a subsequent STAF request from the command line which referenced the previous handle or was dependent upon the existence of the previous handle, your request would fail.

STAF requests submitted from the command line are generally used to query information from STAF services.

Before an application can submit STAF requests, it must first register with STAF. Registering with STAF provides your program with a handle to which your program can submit any number of STAF requests. This handle will remain active in STAF until your program unregisters the handle or until the process ends.

We'll see specific examples of these issues later in this document.

3. STAF Services

3.1. Internal STAF Services

The executable code for internal STAF services resides within STAFProc, which means they are always available and have a fixed name.

DIAG	Provides diagnostics services	*Internal ("DIAG")*
DELAY	Provides a means to sleep a specified amount of time	*Internal ("DELAY")*
ECHO	Echos back a supplied message	*Internal ("ECHO")*
FILE SYSTEM	Allows you to get and copy files across the network	*Internal ("FS")*
HANDLE	Provides information about existing STAF handles	*Internal ("HANDLE")*
HELP	Provides Help on STAF error codes	*Internal ("HELP")*
MISC	Handles miscellaneous commands such as displaying the version of STAF that is currently running	*Internal ("MISC")*
PING	Provides a simple is-alive message	*Internal ("PING")*
PROCESS	Allows you to start, stop, and query processes	*Internal ("PROCESS")*
QUEUE	Provides a network-enabled IPC mechanism for STAF Programs	*Internal ("QUEUE")*
SEMAPHORE	Provides network-enabled named event and mutex semaphores	*Internal ("SEM")*
SERVICE	Allows you to list services available on a machine and to examine the Requests that have been submitted on a machine	*Internal ("SERVICE")*
SHUTDOWN	Provides a means to shutdown STAF and register for shutdown notifications	*Internal ("SHUTDOWN")*
TRACE	Provides tracing information for STAF services	*Internal ("TRACE")*
TRUST	Interfaces with STAF's security	*Internal ("TRUST")*
VARIABLE	Provides a method for maintaining configuration and runtime data (variables)	*Internal ("VAR")*

3.2. External STAF Services

The executable code for external STAF services resides outside of STAFProc, for example in a Java jar file, a C++ DLL file, or a Rexx script file.

External STAF services must be registered via the STAF.cfg configuration file. The name by which the service is known is specified when the service is registered. You can find out more about registering external services later in this document.

Note that you may want to install and register some external STAF services (e.g. STAX, Event, Monitor, ResPool) on just one machine in your STAF test environment. This allows the other STAF machines in your test environment to send requests for these services to that one machine; thus, each machine in the test environment does not have to have these external STAF services installed and registered

CRON	Calls into STAF services at a specified time interval	*External (Java)*
EMAIL	Allows you to send email messages	*External (Java)*
EVENT	Provides a publish/subscribe notification system	*External (Java)*
EVENTMANAGER	Allows you to call STAF services when a specified Event occurs	*External (Java)*
HTTP	Allows you to make HTTP requests which can be grouped together in a session	*External (Java)*
LOG	Provides a full-featured logging facility	*External (C++)*

MONITOR	Allows a testcase to publish its current running execution status for others to read	*External (C++)*
RESOURCE POOL	Allows you to manage exclusive access to pools of elements, e.g. VM UserIDs or Software Licenses	*External (C++)*
STAX	Provides an XML-based execution engine	*External (Java)*
ZIP	Provides a means to zip/unzip/list/delete PKZip/WinZip compatible archives	*External (C++)*

3.3. Delegated STAF Services

STAF services may also be delegated to another machine in the STAF environment. In this case, when a request is made for the service on the local STAF machine, it is automatically forwarded to the machine to which this service has been delegated.

3.4. STAF ServiceLoaders

STAF ServiceLoaders are external services whose purpose is to load services on-demand. They allow services to be loaded only when they have been requested, so they don't take up memory until needed. They also allow dynamic service registration when a request is made so that you don't have to change the STAF configuration file to register a service.

When a request is encountered for a service that doesn't exist, STAF will call each serviceloader, in the order they were configured, until the service exists or we run out of servicelaoders. If we run out of serviceloaders, then the standard RC:2 will be returned. Otherwise, the request will be sent to the newly added service. A default serviceloader is shipped with STAF, and it can dynamically load the Log, Monitor, ResPool, and Zip services.

3.5. Custom STAF Services

Note that you can also write your own custom services that can be plugged into STAF. These services can be written in Java or C++.

3.6. STAF Authenticators

Authenticators are special external services whose purpose is to authenticate users in order to provide user level trust, which can be used in addition (or instead of) machine level trust. An Authenticator is a special service that accepts an authenticate request. As a user, you cannot directly submit a request to an authenticator service. Authenticators are accessed indirectly via the Handle service.

3.7. STAF Connection Providers

Currently, the only network interface which comes with STAF is TCP/IP. However, STAF allows you to plug in network interfaces, called Connection Providers, so that you can create your own connection provider which can communicate via any mechanism you choose (e.g. SSL, a Serial Line, NetBIOS, or SNA). Connection provider interfaces are C/C++ based so they are platform specific.

The STAF TCP/IP Connection Provider supports both IPv4 and IPv6.

4. STAF Commands

Now that you've installed STAF and understand some of the basic concepts, it's time to actually start using STAF. We suggest that as you go through this tutorial, you work with 2 machines that both have STAF installed. This will allow you to submit STAF requests not only locally, but to remote machines. This will be especially beneficial during the STAF demo and will show you the capabilities and power of STAF.

In this tutorial, the images and commands shown will reference machines "staf3c" (our local machine) and "staf1c" (our remote machine). You will need to substitute your machines names.

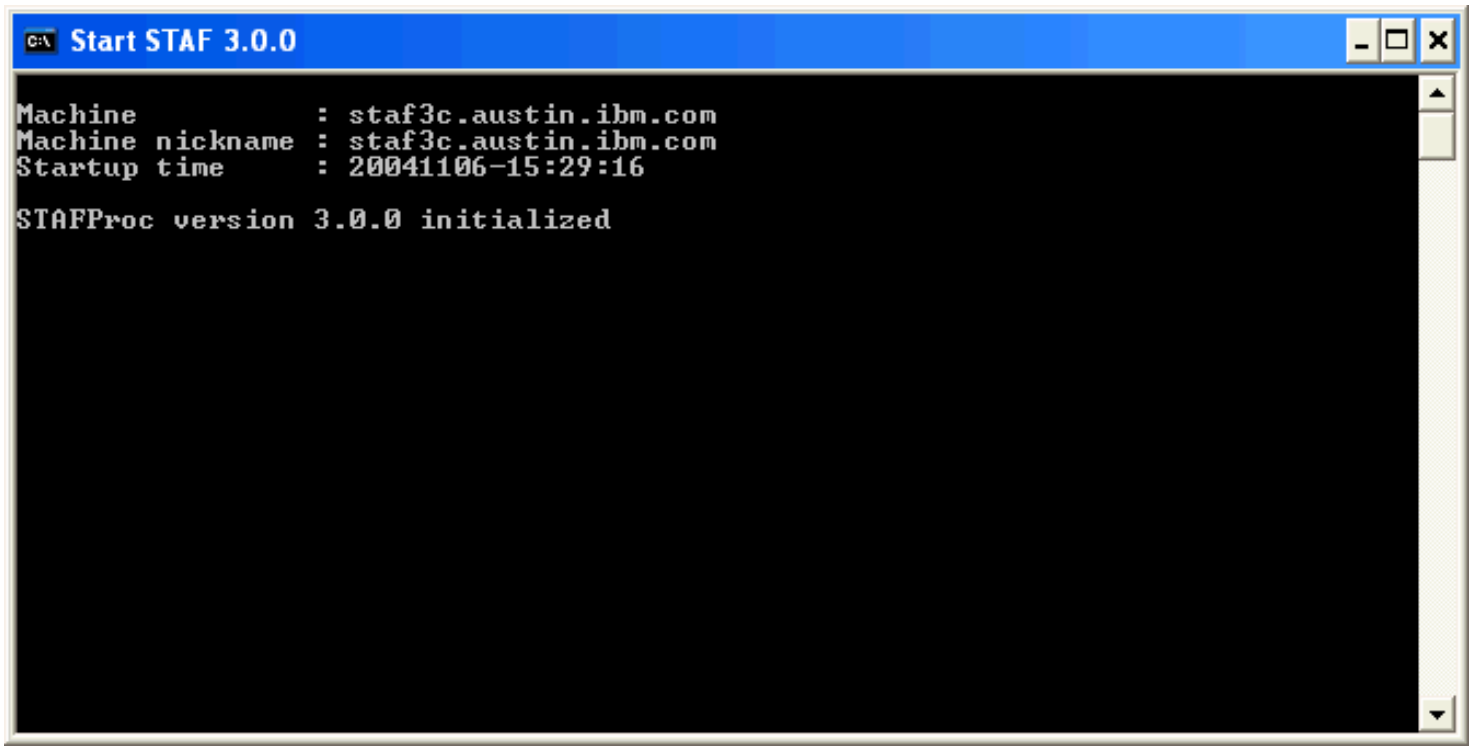
4.1. Starting STAFProc

The STAFProc command is what starts the STAF daemon process running on a machine. On Windows machines, you can also start STAFProc via the Start menu (just go to the folder where you chose to install STAF, titled "STAF 3.0.0" and click on "Start STAF". Note that on Unix systems you will need to ensure that /usr/local/staf/bin is in your PATH).

You can also start STAFProc by simply typing STAFProc at a command prompt window.

You should see a window similar to Figure 2.

Figure 2.



Of course, the "Machine" and "Machine nickname" would be specific to your system.

If any errors are encountered while STAFProc is starting, error messages will be displayed in this window.

Note that on Windows systems, if you chose to start STAF from the Start menu, and a fatal error is encountered while starting STAF, the "Start STAF" window will close so you will not be able to see the error message. If this occurs, start STAFProc from a command prompt window so that you can see the error messages.

4.2. Shutting Down STAFProc

When shutting down STAF, it is recommended that you always use the SHUTDOWN command of the SHUTDOWN service (or the "Shutdown STAF" program on Windows via the Start menu) rather than just Ctrl-C stopping STAFProc. This will allow STAF to free any resources which may be in use.

The command to issue is:

```
STAF local shutdown shutdown
```

After the shutdown completes, you should see the message "STAFProc ending normally" and STAFProc should then terminate.

4.3. Submitting STAF Requests from the Command Line

Notice that in the above shutdown command, the program executed is STAF. This is an executable that is used to submit requests to STAF from the command line.

The syntax of this command is:

```
STAF <Endpoint> <Service> <Request>
```

<Endpoint> is either LOCAL, if you wish to make a request of the local machine, or the name of the machine of which you wish to make a request

<Service> is the name of the service that will receive and process the request

<Request> is the service request

The STAF command line utility works just like any other STAF application. It registers with STAF, performs a request (which is the service request you specify), and then unregisters. That last step causes the handle to be deleted. This somewhat limits the usage of the STAF command line utility.

When using STAF from batch or shell scripts, see section 5.2 of the STAF User's Guide for information on working around these limitations.

4.4. Remote system identification

When making a STAF request to a remote system, in addition to specifying the machine name, you may also specify the network interface over which communication will take place. The format for this is

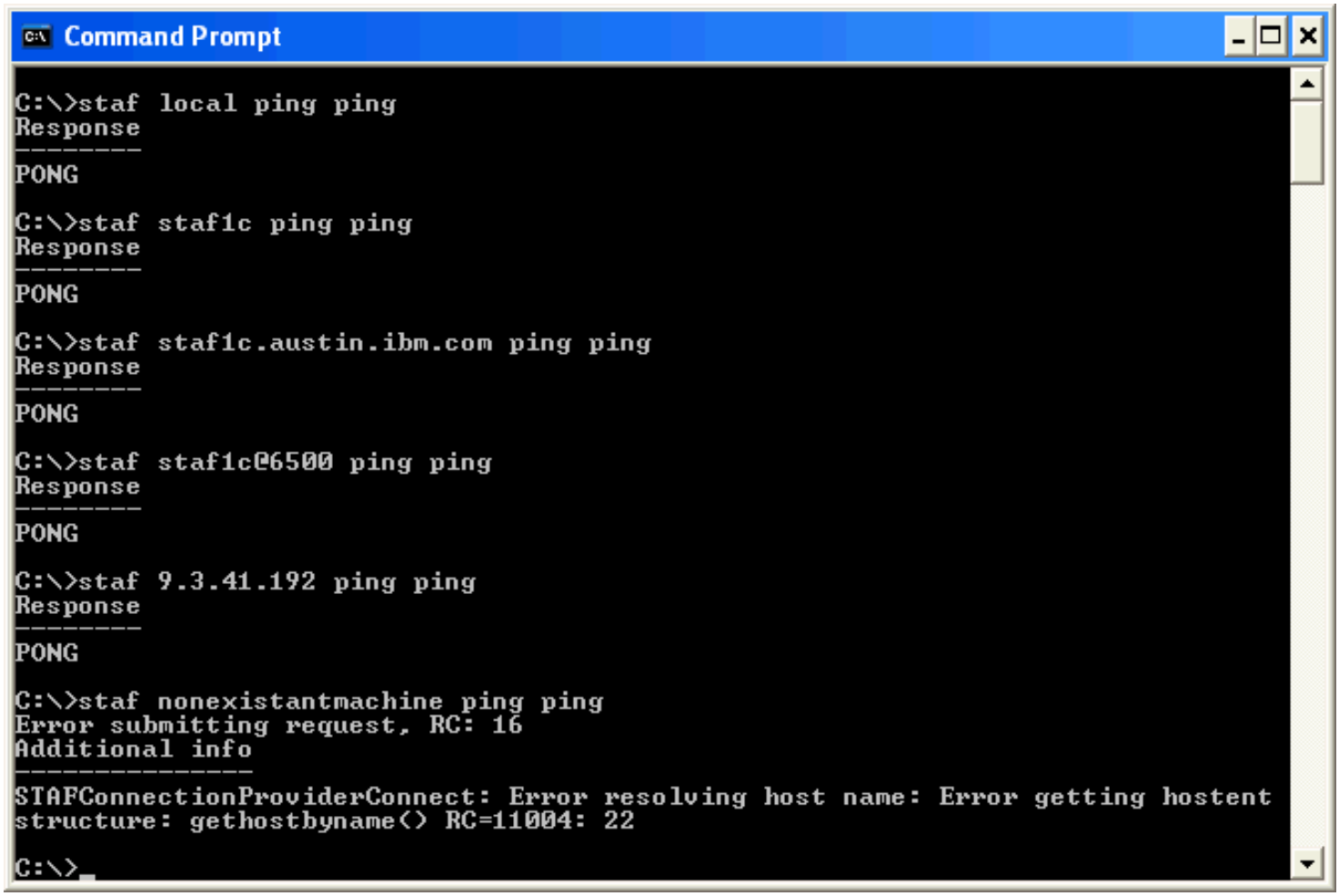
```
[<Interface>://]<System Identifier>[@<Port>]
```

where **<Interface>** is the name of the network interface and **<System Identifier>** is a valid network identifier for the **<Interface>** in question. If no **<Interface>** is specified, the default interface is used. You may specify logical or physical identifiers. For example, for a TCP/IP interface, the physical identifier for a system is the IP address, while the logical identifier is the hostname. You may optionally specify a valid port to use for a TCP/IP interface.

4.5. Pinging Machines

To make certain that you can access a machine via STAF (and that the machine is alive), you can use the PING service as follows

Figure 3.



The first command, **`STAF local ping ping`**, simply pings the local machine.

The second command, **`STAF staf1c ping ping`**, demonstrates that you can also use the short hostname (staf1c) for the remote machine.

The third command, **`STAF staf1c.austin.ibm.com ping ping`** pings a remote machine using its full hostname (staf1c.austin.ibm.com).

The fourth command, **`STAF staf1c@6500 ping ping`** pings a remote machine using its short hostname on TCP/IP port 6500.

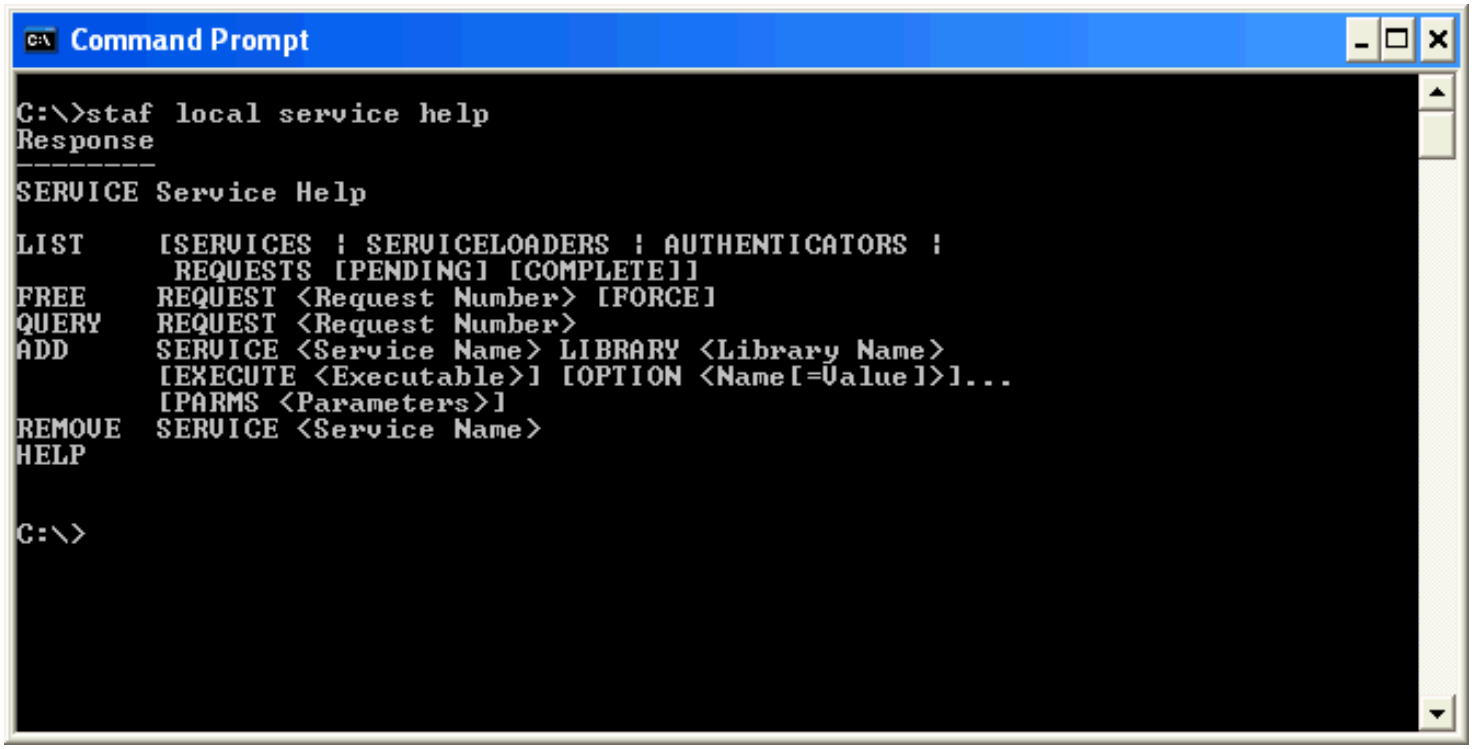
The fifth command, **`STAF 9.3.41.192 ping ping`** demonstrates that you can use IP addresses.

The sixth command, **`STAF nonexistantmachine ping ping`**, fails because remote machine nonexistantmachine cannot be found.

4.6. Obtaining Help for a Service

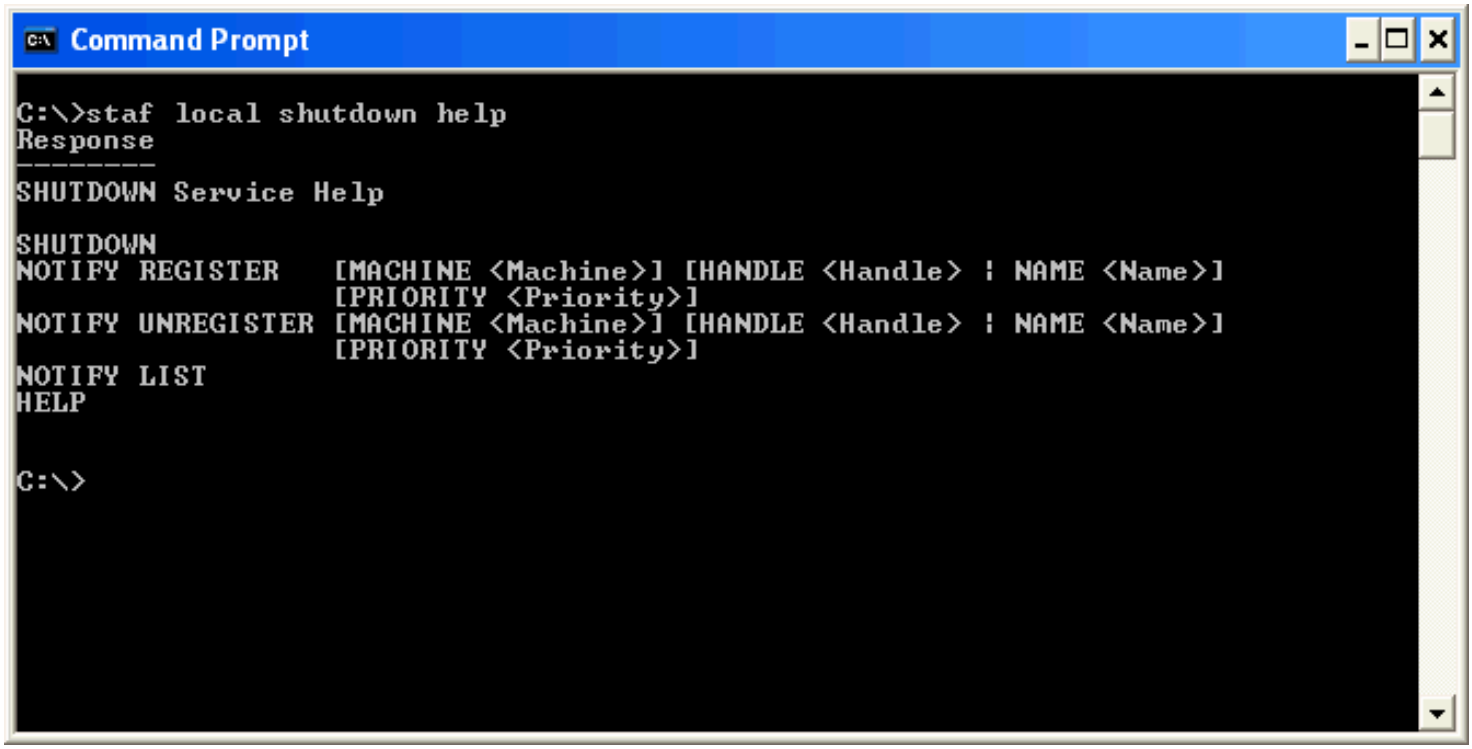
To obtain help for a service, issue the following command: **`STAF local service help`** This returns the valid service request strings; the result should look like:

Figure 4.



In this case we are actually requesting help for the "service" service. To request help for another service, just change "service" to the other service name. For example, to obtain help for the "shutdown" service, type: **`STAF local shutdown help`** The result should look like:

Figure 5.

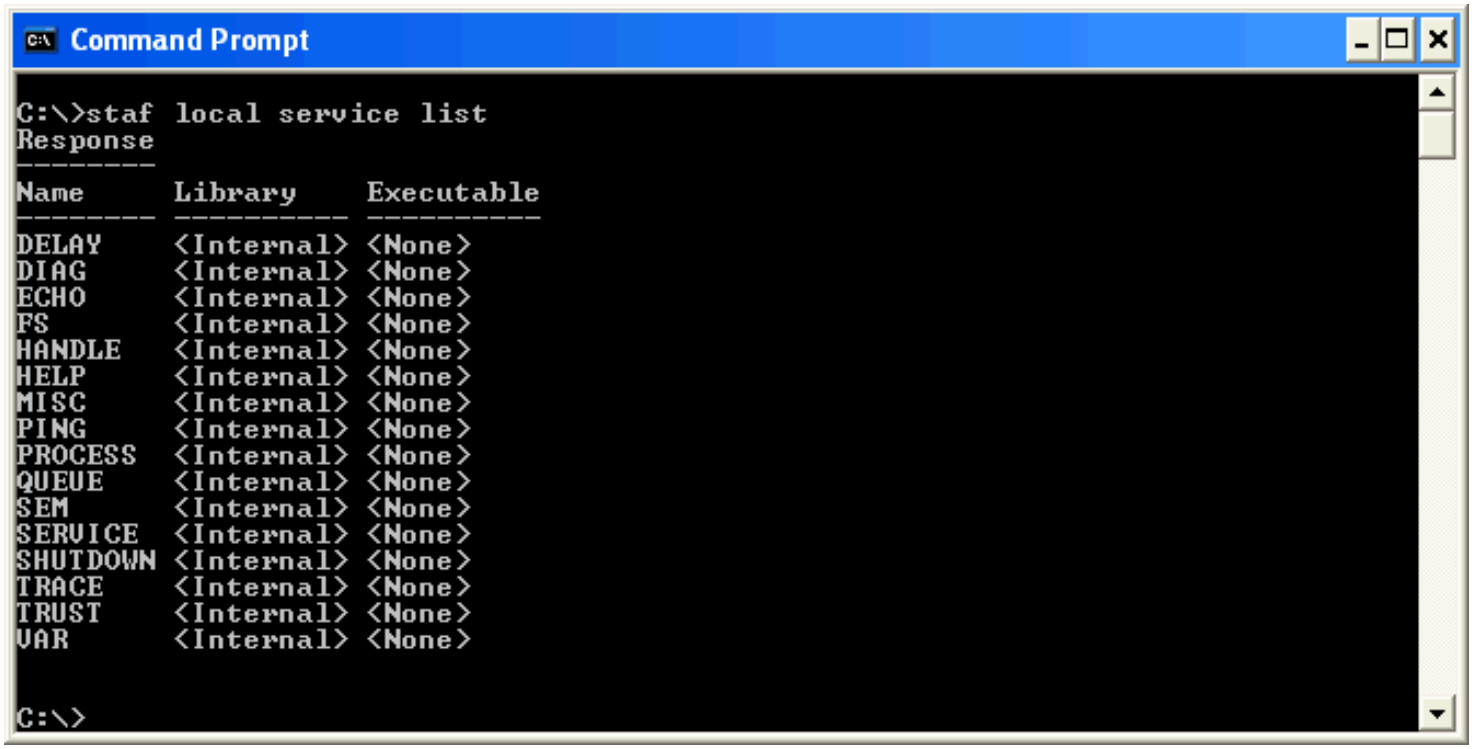


Notice that two of the commands returned were "SHUTDOWN" and "HELP". The information returned by Help show us the options we can place after "STAF local shutdown ....." in command requests for the Shutdown service.

4.7. Listing Available Services

To list available STAF services, issue the following command from a command prompt: **`STAF local service list`**

Figure 6.

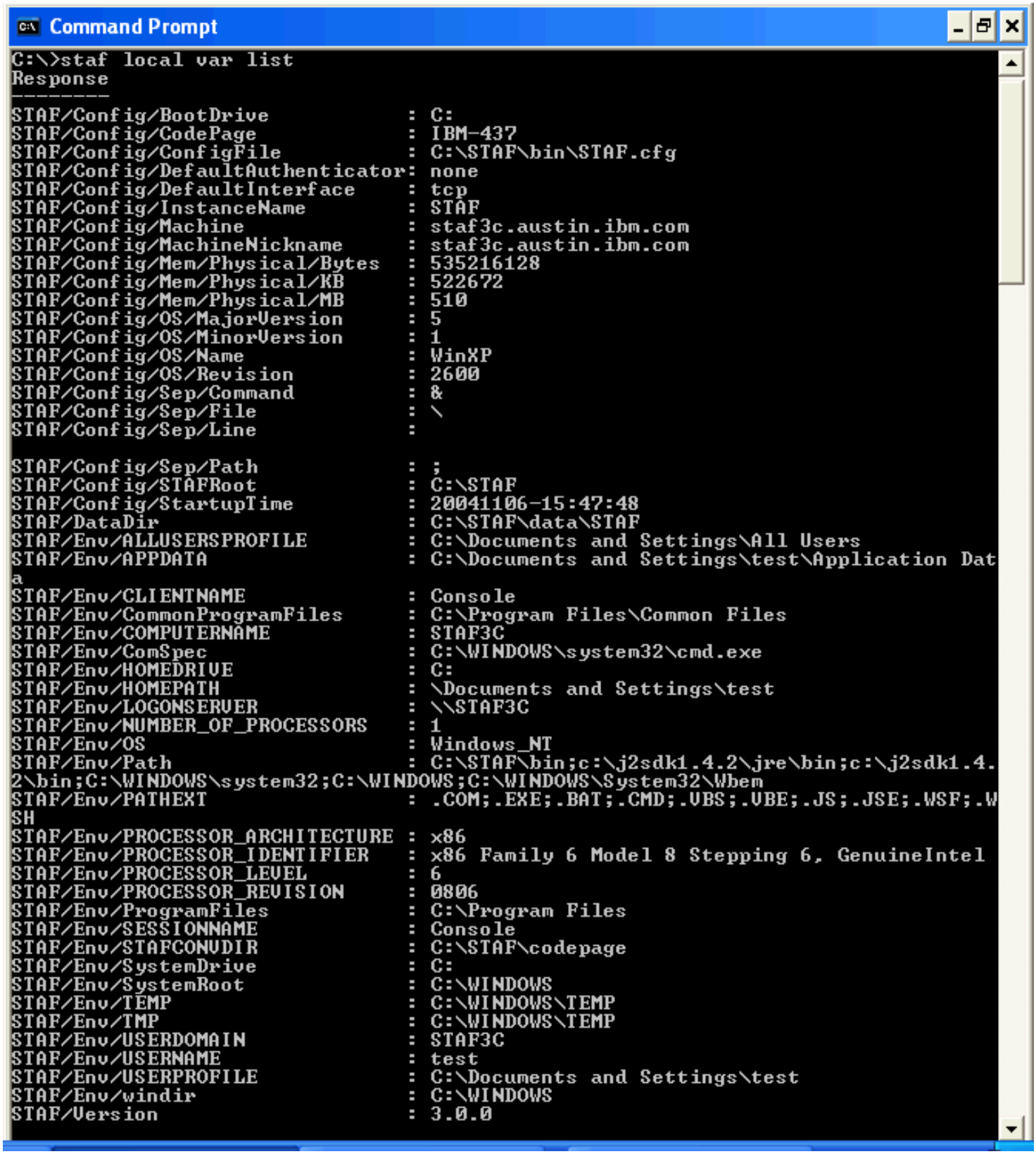


Notice in the response that only internal services are available. This is because we have not yet registered any external services in the STAF configuration file.

4.8. Listing Variables

To list available STAF variables, issue the following command from a command prompt: **`STAF local var list`**

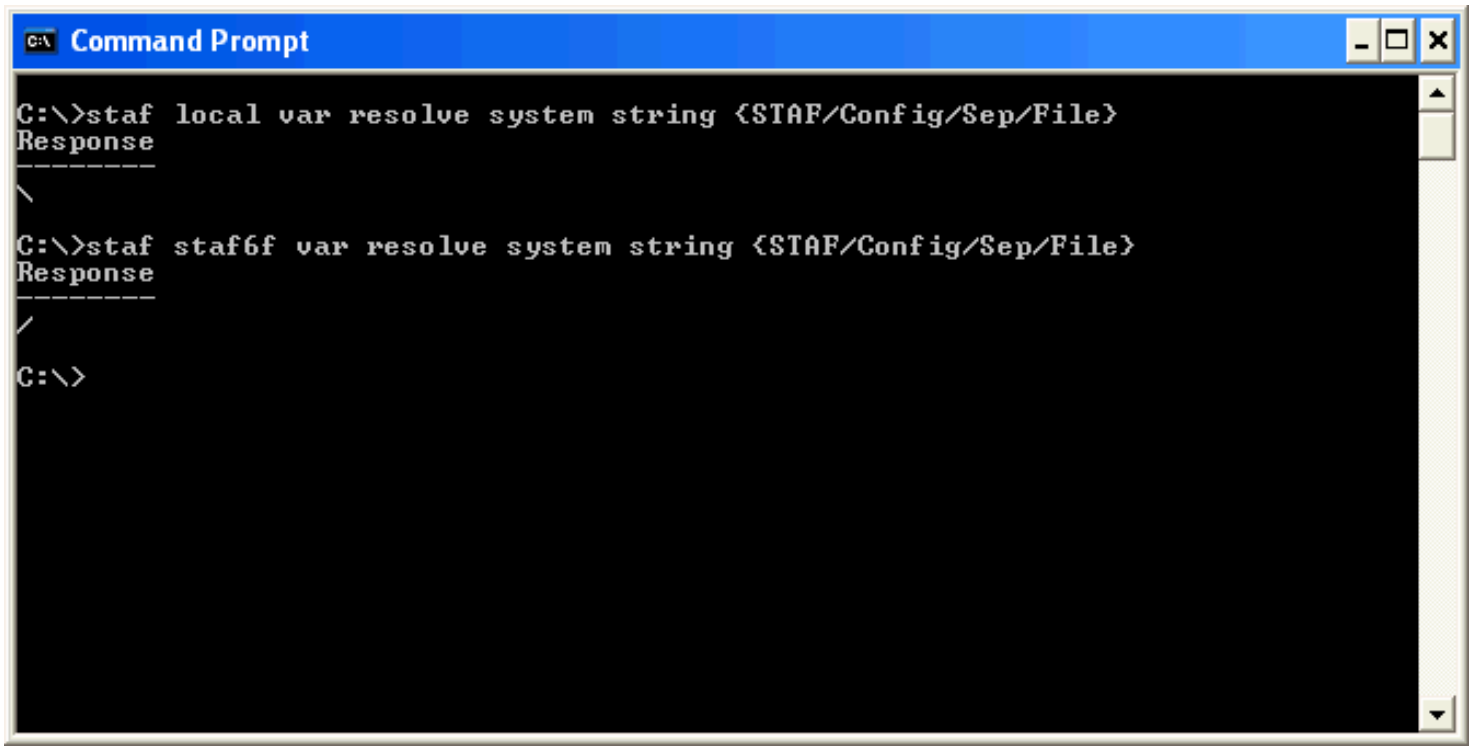
Figure 7.



This lists all of the STAF variables. Notice that even though we have not yet defined any variables in the STAF configuration file, STAF predefines many useful variables, including information about the machine's Operating System and File/Line/Path separators.

To get the value of a specific system variable, for example the file separator, issue the following command (note that the local machine in this example is running Windows): **`staf local var resolve system string {STAF/Config/Sep/File}`**

Figure 8.

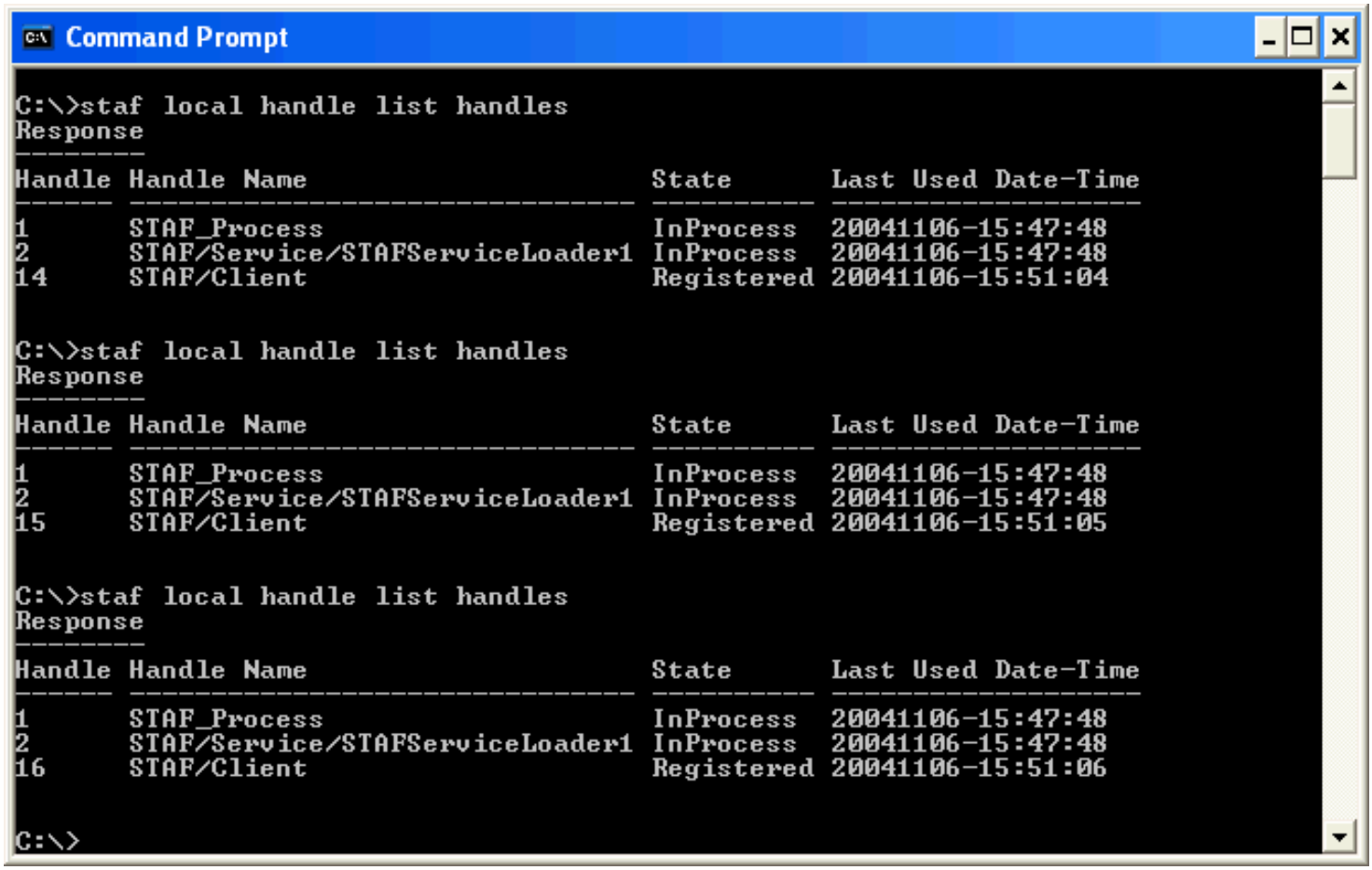


Notice that the second command **`staf staf6f var resolve system string {STAF/Config/Sep/File}`** was sent to a machine running a Unix operating system, so the output reflects the Unix File Separator.

4.9. Listing Handles

To list the current STAF handles, issue the following command: **`STAF local handle list handles`**

Figure 9.



Notice that in each response above, handle 1 is assigned to STAFProc. Each of the STAF/Client requests represent each of the three "STAF local handle list handles" commands you submitted. Note that each request is assigned a new handle number, and that the previous handles have been deleted (for example, the third response does not show handles 14 and 15).

5. Configuring STAF

5.1. STAF Configuration File

STAF is configured through a text file called the STAF configuration file. This file may have any name you desire, but the default is STAF.cfg. If you want to use a different name for the file, then this name must be passed as the first parameter when starting STAFProc. This file is located in the c:\staf\bin directory on Windows systems, or /usr/local/staf/bin on Unix systems.

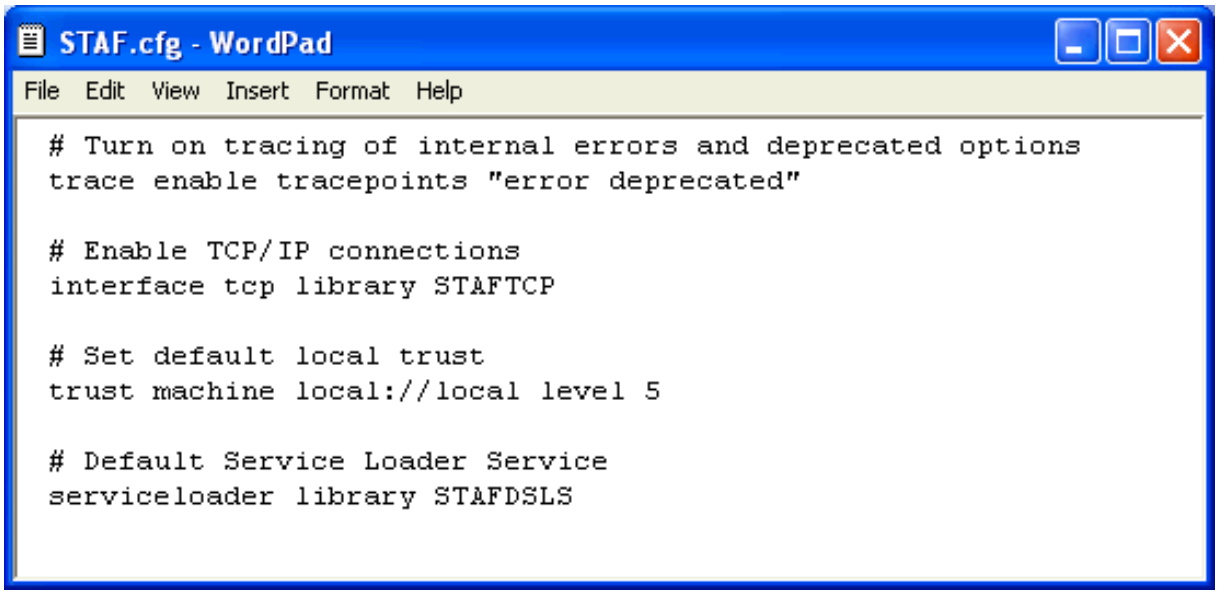
When STAFProc is started on a machine, that machine's STAF.cfg file will be read to determine how STAF should be configured on the machine.

Note that whenever you make changes to the STAF.cfg file, you must restart STAFProc in order for the modified configuration information to be read.

5.2. Default STAF.cfg

When you first install STAF, a default STAF.cfg file will be created for you. Here is what the default file looks like:

Figure 10.



Notice that comments start with **#**.

The first configuration statement is **`trace enable tracepoints "error deprecated"`**. This statement causes a trace message to be generated for error conditions that STAF detects, such as broken communication connections and fatal service errors, as well as for deprecated options that STAF detects.

The second configuration statement is **`interface tcp library STAFTCP`**. This statement is used to indicate that you wish to send and accept STAF requests on a network interface. The default port which is used by STAF is 6500. If you wish to specify a port other than 6500, you would specify the port number at the end of the statement. For example, to use port 6600, the statement would be **`interface tcp2 LIBRARY STAFTCP OPTION PORT=6600`**.

The third configuration statment is **`trust machine local://local level 5`**, which sets the local trust level to 5 (full access).

The fourth configuration statement, **`serviceloader Library STAFDSLS`** registers the default ServiceLoader, which can dynamically load the Log, Monitor, ResPool, and Zip services.

You can see that there isn't much to the default STAF.cfg. Now we'll start adding statements to it.

5.3. Machine Nickname

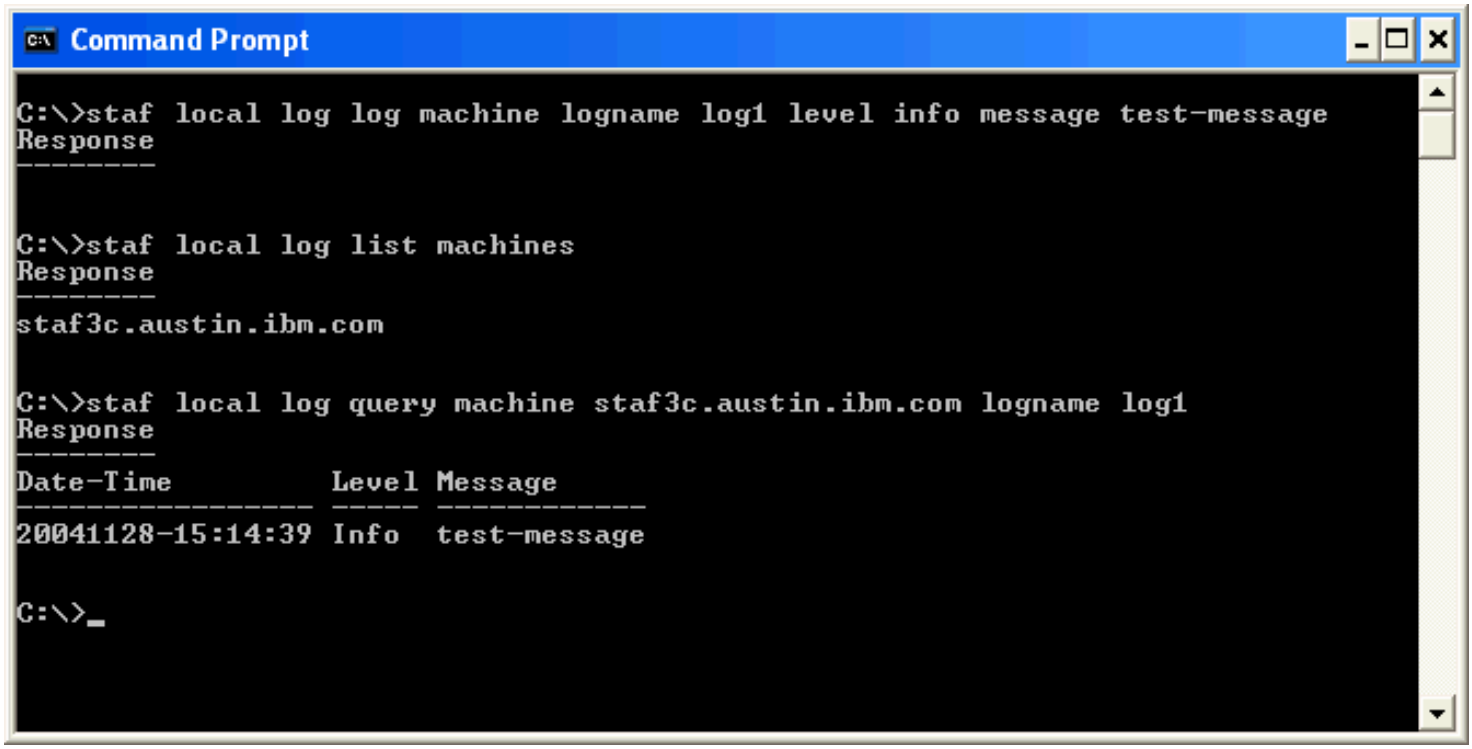
You may specify a nickname for your machine using the MACHINENICKNAME configuration statement. This overrides the value of the STAF/Config/MachineNickname system variable. This primarily effects the data stored by services such as the Log and Monitor services, which store data based on the machine from which it came by using the STAF/Config/MachineNickname system variable as part of the directory path when creating logs and monitor data. By allowing the STAF/Config/MachineNickname system variable to be overridden, it allows you to better manage your data.

Note that the machine nickname is not used to communicate with other systems and does not have any effect on trust.

Run the following commands:

```
staf local log log machine logname log1 level info message test-message
staf local log list machines
staf local log query machine <your-long-hostname> logname log1
```

Figure 11.



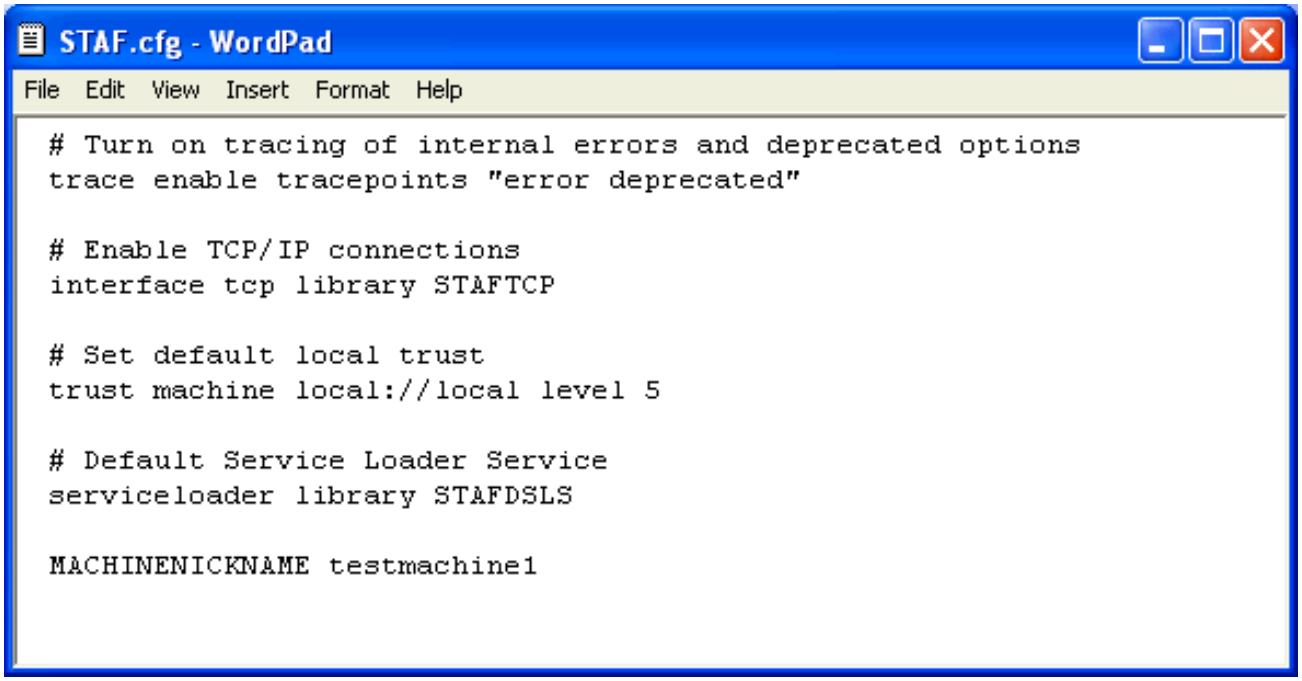
Notice that by default the "machine" that the Log service uses to store machine logs is the long hostname for the machine.

Now add the following statement to your STAF.cfg file:

```
MACHINENICKNAME testmachine1
```

Add this statement to your STAF.cfg file and save it. Remember to shutdown and restart STAFProc to pick up the STAF.cfg updates.

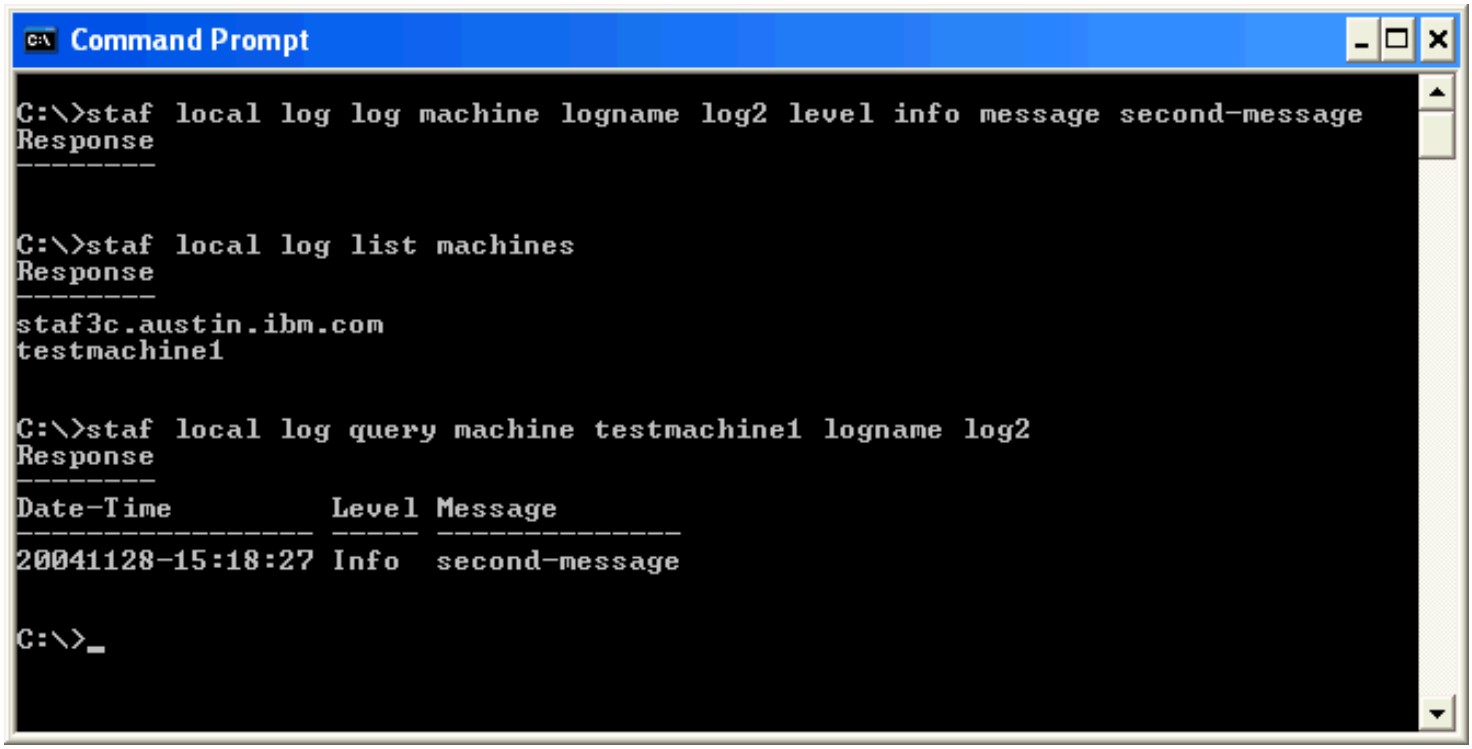
Figure 12.



Now run the following commands:

```
staf local log log machine logname log2 level info message second-message
staf local log list machines
staf local log query machine testmachine1 logname log2
```

Figure 13.



Notice that now the Log service is using the machine nickname as the "machine" that the Log service uses to store machine logs.

5.4. Operational Parameters

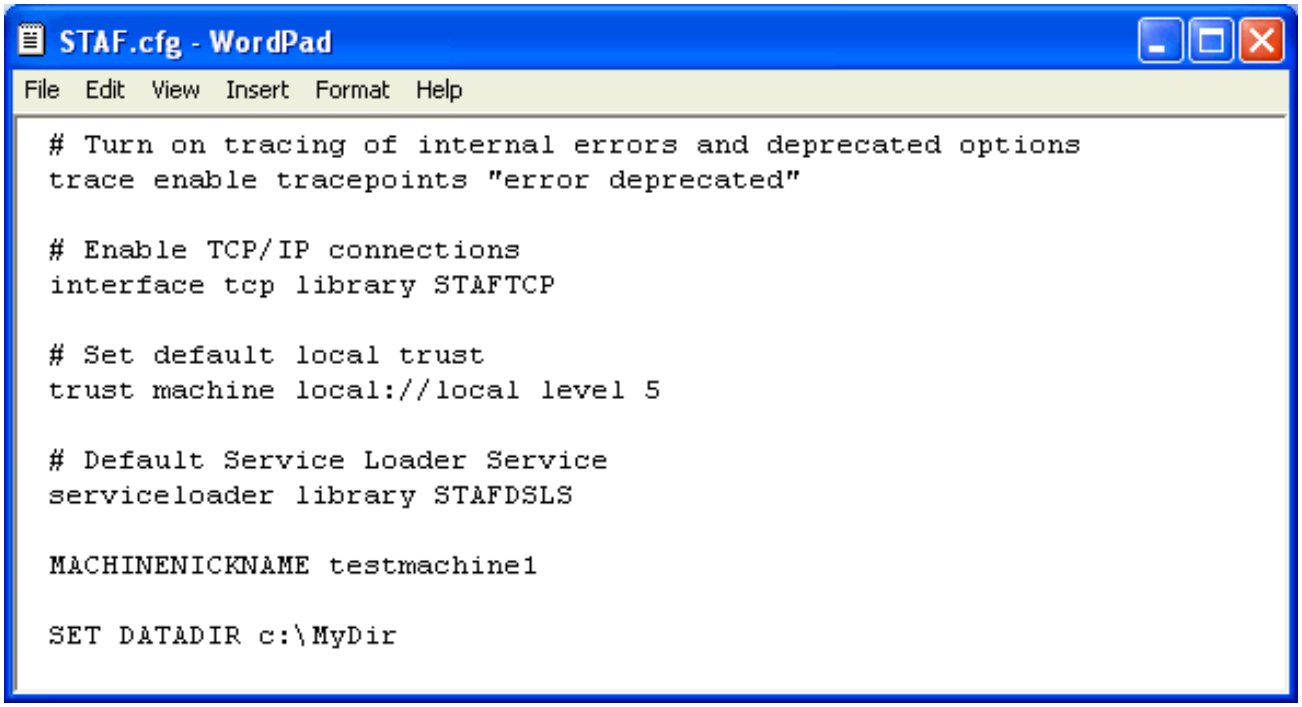
Through the SET command, STAF allows you to set various parameters which affect the general operation of STAF. The STAF User's Guide lists all of the available parameters.

One of the available parameters is DATADIR. It specifies the directory that STAF and its services will use to write data. This allows STAF to be installed in a location that is read-only when STAFProc is running. By default the directory will be {STAF/Config/STAFRoot}/data/{STAF/Config/InstanceName}. To use a different directory, use the following statement:

```
SET DATADIR C:\MyDir
```

Add this statement to your STAF.cfg file and save it. Remember to shutdown and restart STAFProc to pick up the STAF.cfg updates.

Figure 14.

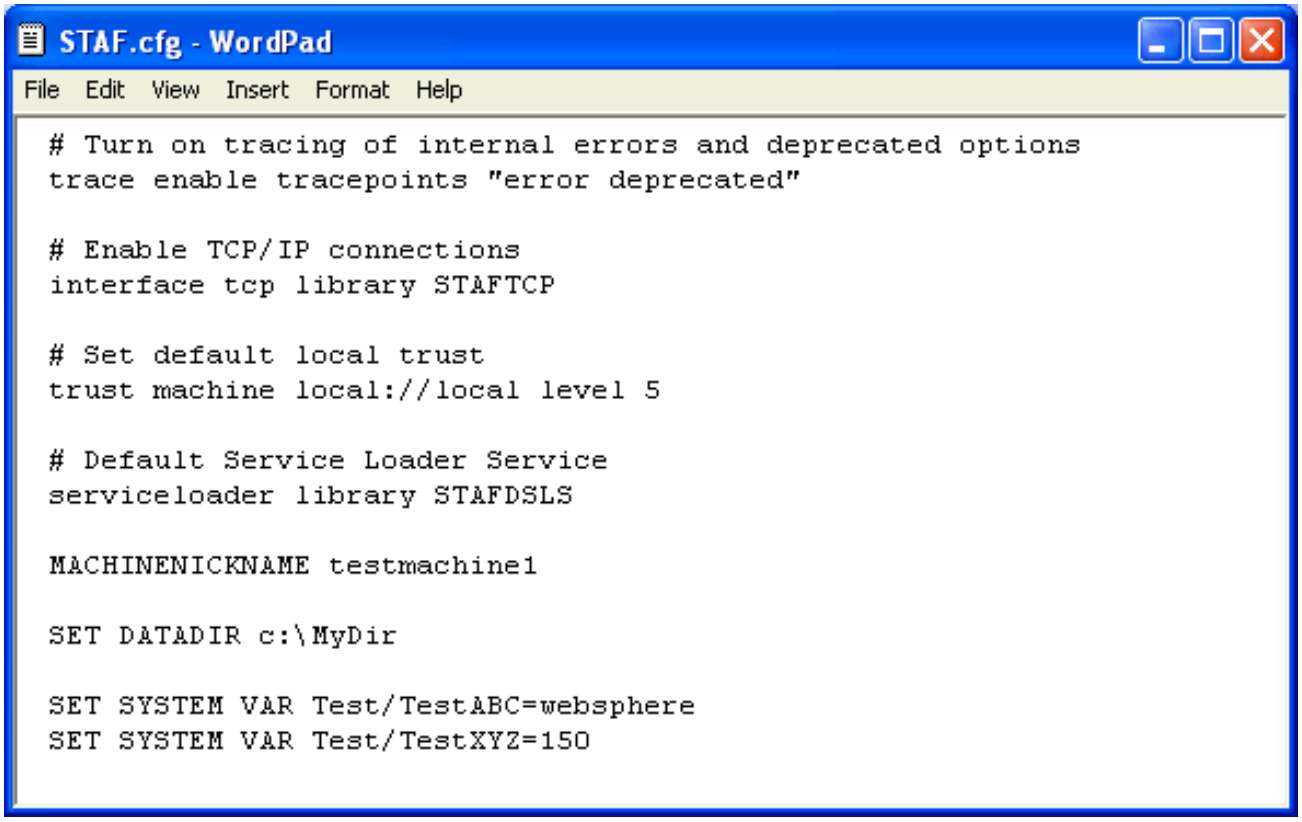


5.5. Setting Variables

You may set system STAF variables at startup by using the SET VAR configuration statement. Add the following test variables to your STAF.cfg file:

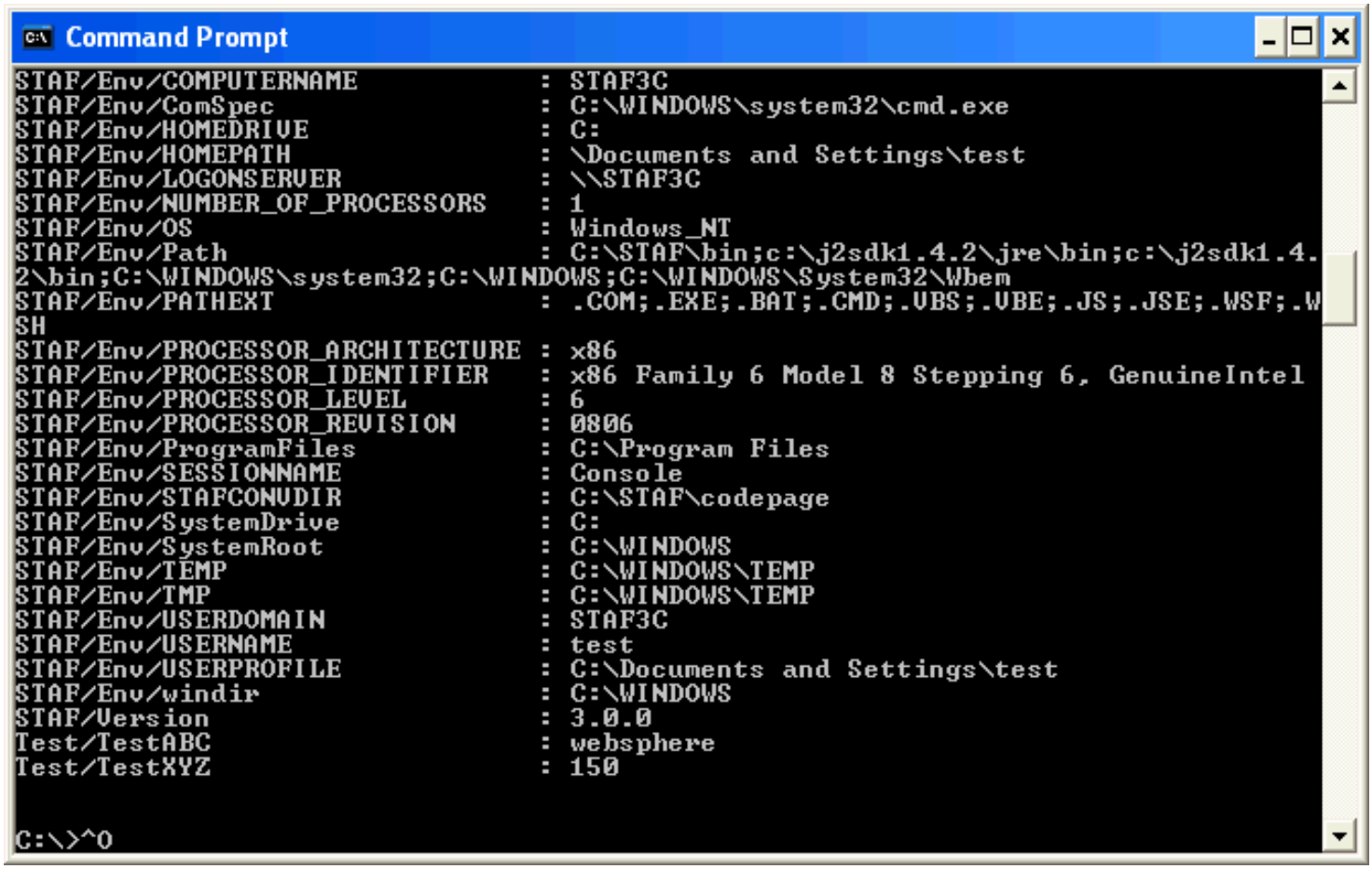
```
SET SYSTEM VAR Test/TestABC=websphere
SET SYSTEM VAR Test/TestXYZ=150
```

Figure 15.



Now restart STAFProc and from a command prompt, try the **`STAF local var list`** command:

Figure 16.



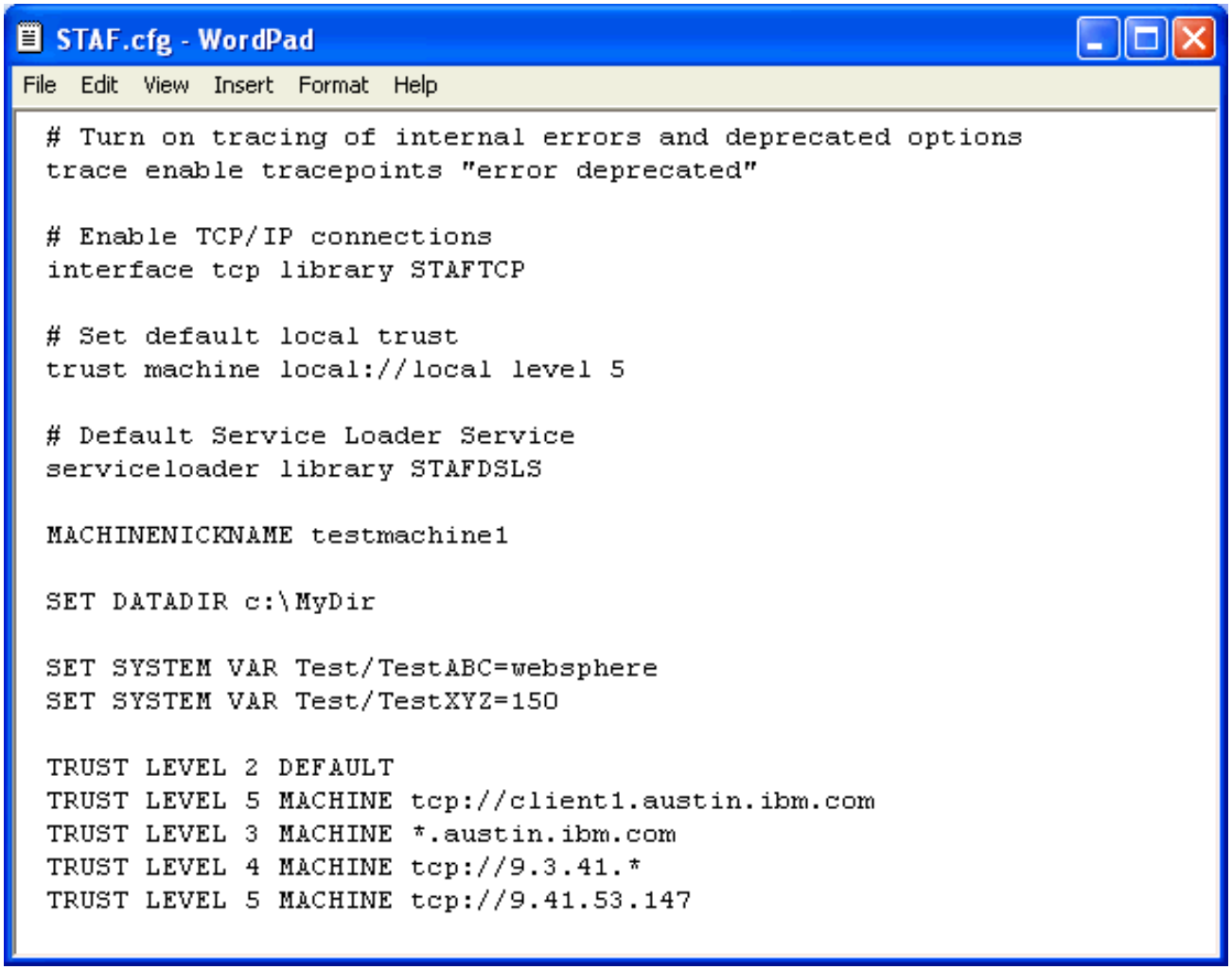
Notice that the 2 test variables are now included at the bottom of the output.

5.6. Trust Levels

STAF allows you to grant access to machines and users by using the TRUST configuration statement. Add the following statements to your STAF.cfg:

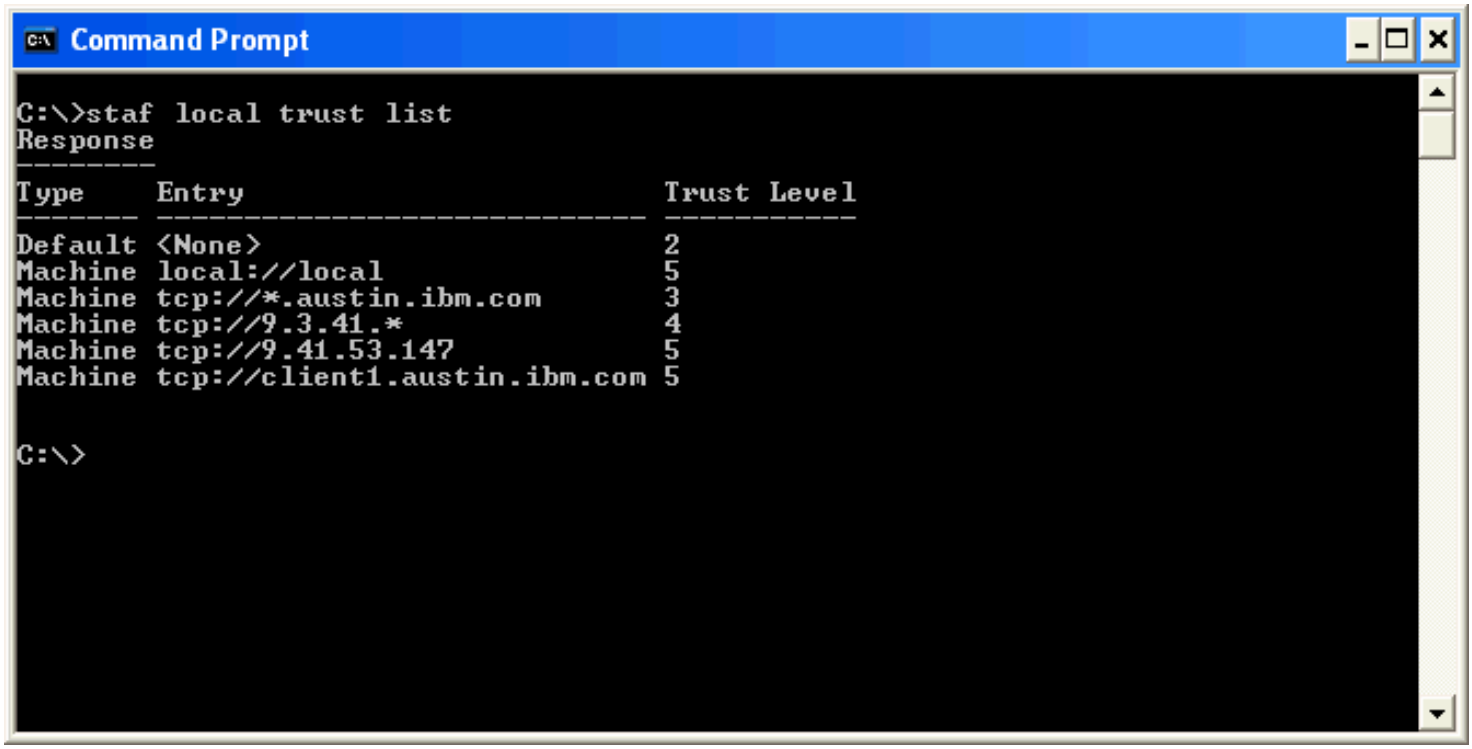
```
TRUST LEVEL 2 DEFAULT
TRUST LEVEL 5 MACHINE tcp://client1.austin.ibm.com
TRUST LEVEL 3 MACHINE *.austin.ibm.com
TRUST LEVEL 4 MACHINE tcp://9.3.41.*
TRUST LEVEL 5 MACHINE tcP://9.41.53.147
```

Figure 17.



The numeric trust levels are defined in the STAF User's Guide; higher numbers indicate greater access. Now restart STAFProc on your machine, and run the command **`staf local trust list`**.

Figure 18.



Setting the default trust level to 2 indicates that all machines in the STAF Environment that are not specified in other TRUST configuration statements will have a trust level of 2. If you do not specify a default trust level in your STAF.cfg file, the default is set to 3.

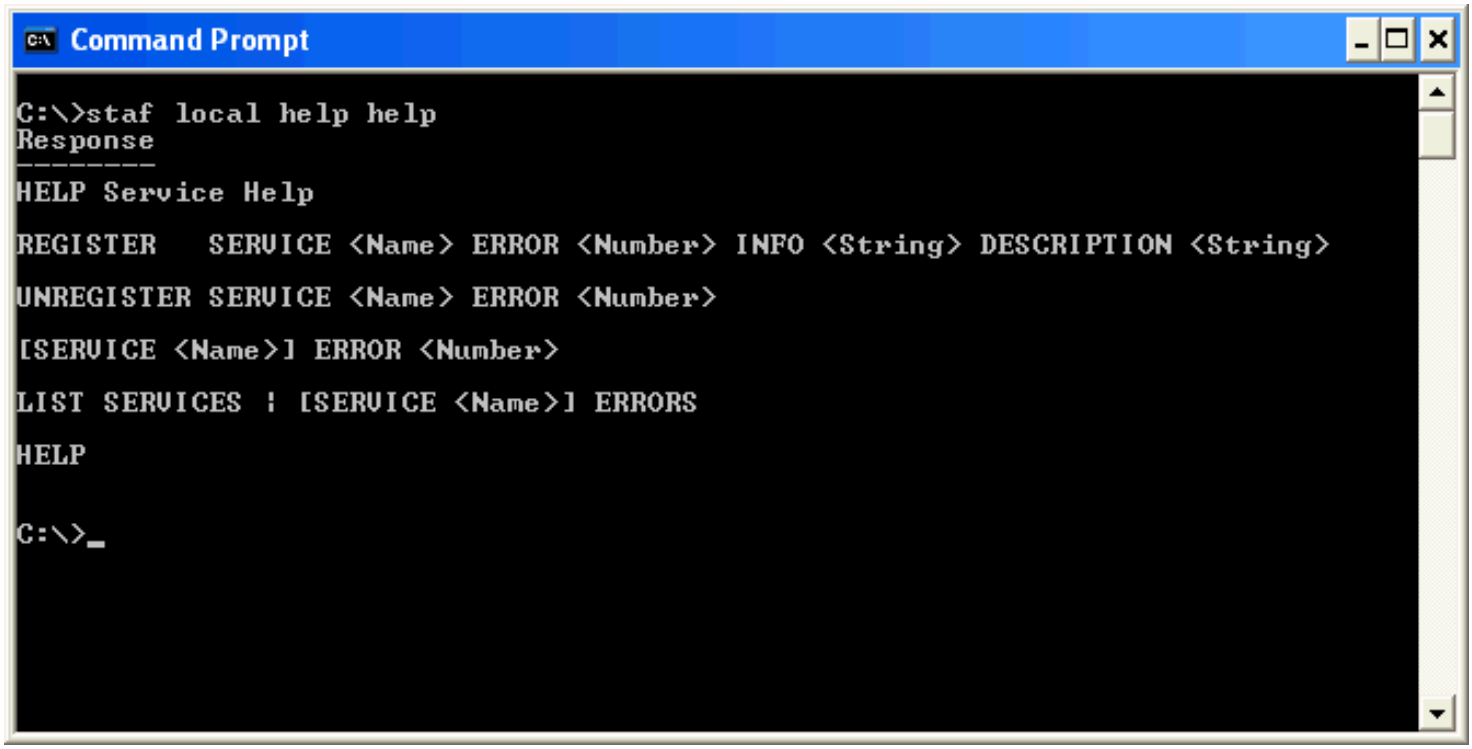
Remember that all STAF services define trust levels for each of the requests that they accept. These trust level requirements for the STAF services are defined in the STAF User's Guide.

Now remove these 5 TRUST entries from your STAF.cfg file and restart STAFProc.

6. Using the Help Service

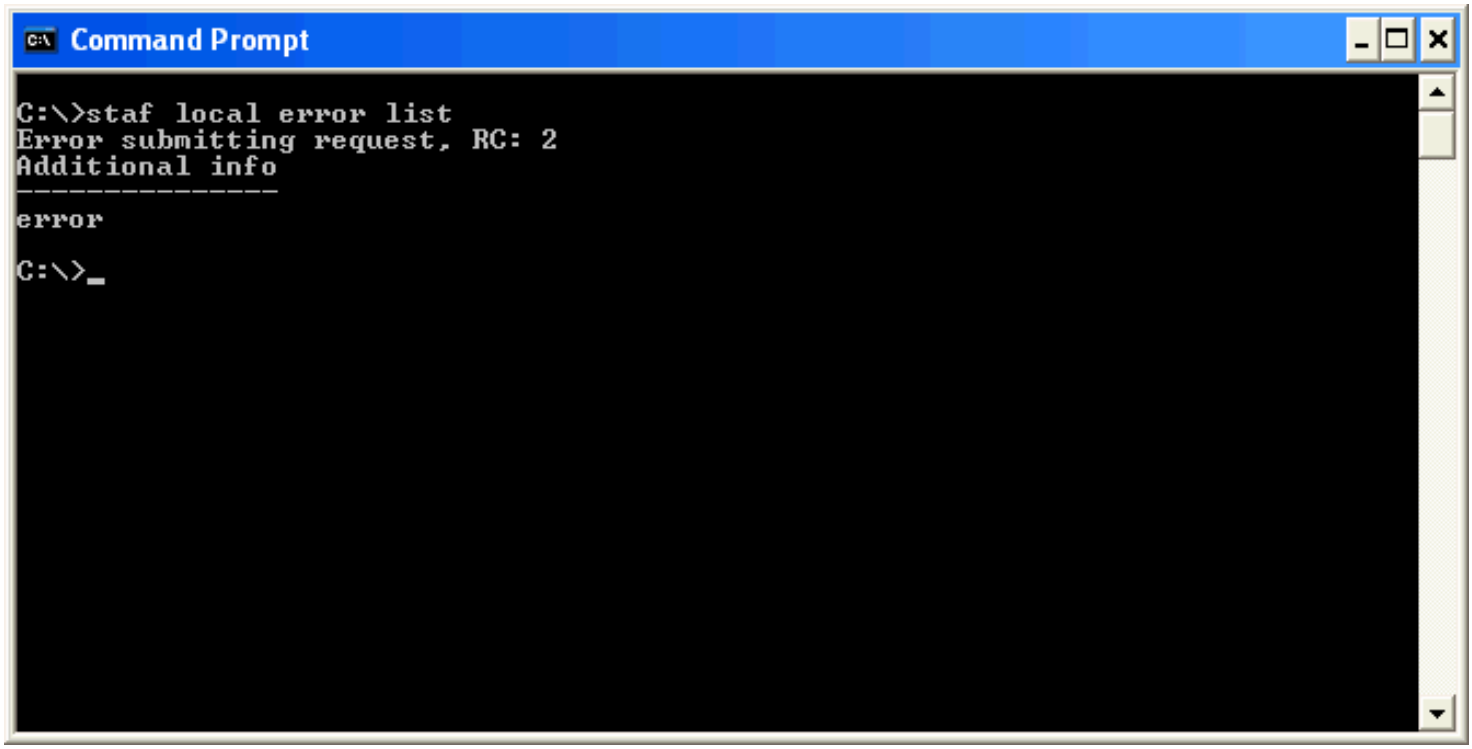
One of STAF's Internal Services, the Help Service, can be very useful when debugging STAF problems. Let's explore the Help service and the information it provides. Issue the following command: **`staf local help help`**

Figure 19.



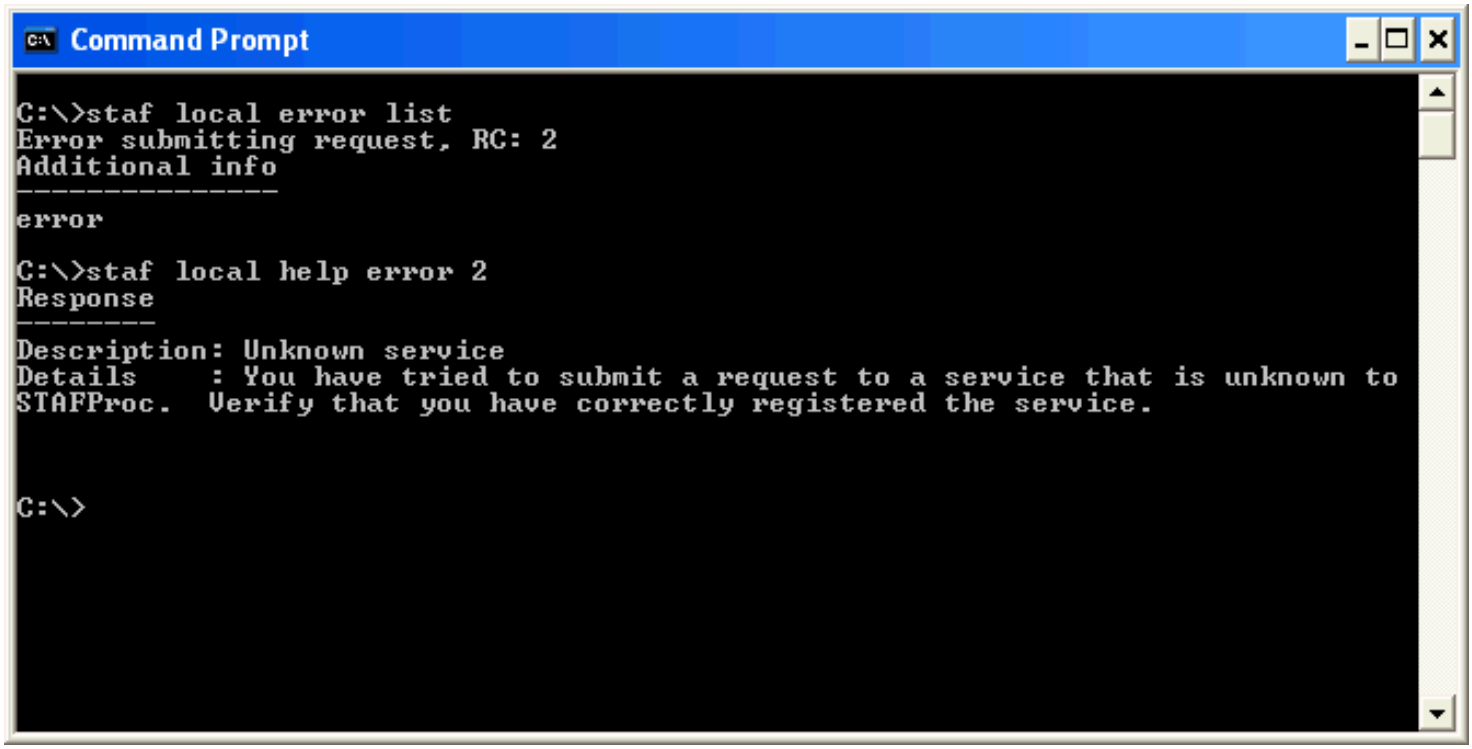
To demonstrate the STAF Help Service, type the following command (note that we are intentionally specifying an invalid service name "error"): **`staf local error list`**

Figure 20.



To find out what a RC 2 means, issue the following command: **`staf local help error 2`**

Figure 21.



The error message explains that there is no "error" service

7. Registering STAF Services

When we issued the **`STAF local service list command`** in the "STAF Commands" section, the only services listed were the Internal services. Now we'll register some External services.

7.1. Using Java STAF Services

In order to run any Java STAF Services and the STAF Demo itself, you will need to have a Java Software Development Kit (SDK) version 1.2 or later, or a Java Runtime Environment (JRE) version 1.2 or later, installed on your test machines.

You can obtain the Sun Java Standard Edition versions from the Sun Java web site.

You can download the external IBM Java versions from the IBM Java web site.

IBM Employees can obtain the IBM Java versions from the IBM Intranet at: IBM Internal Java web site.

After you have installed Java on your test machines, you can verify that the SDK/JRE is set up correctly by typing **`java -version`** from a Command Prompt. The response should be the version of Java you have installed. Note that c:\jdk1.4.2\jre\bin (assuming that you installed Java 1.4.2 to directory c:\jdk1.4.2) must be in your System PATH.

7.2. Registering Java STAF Services

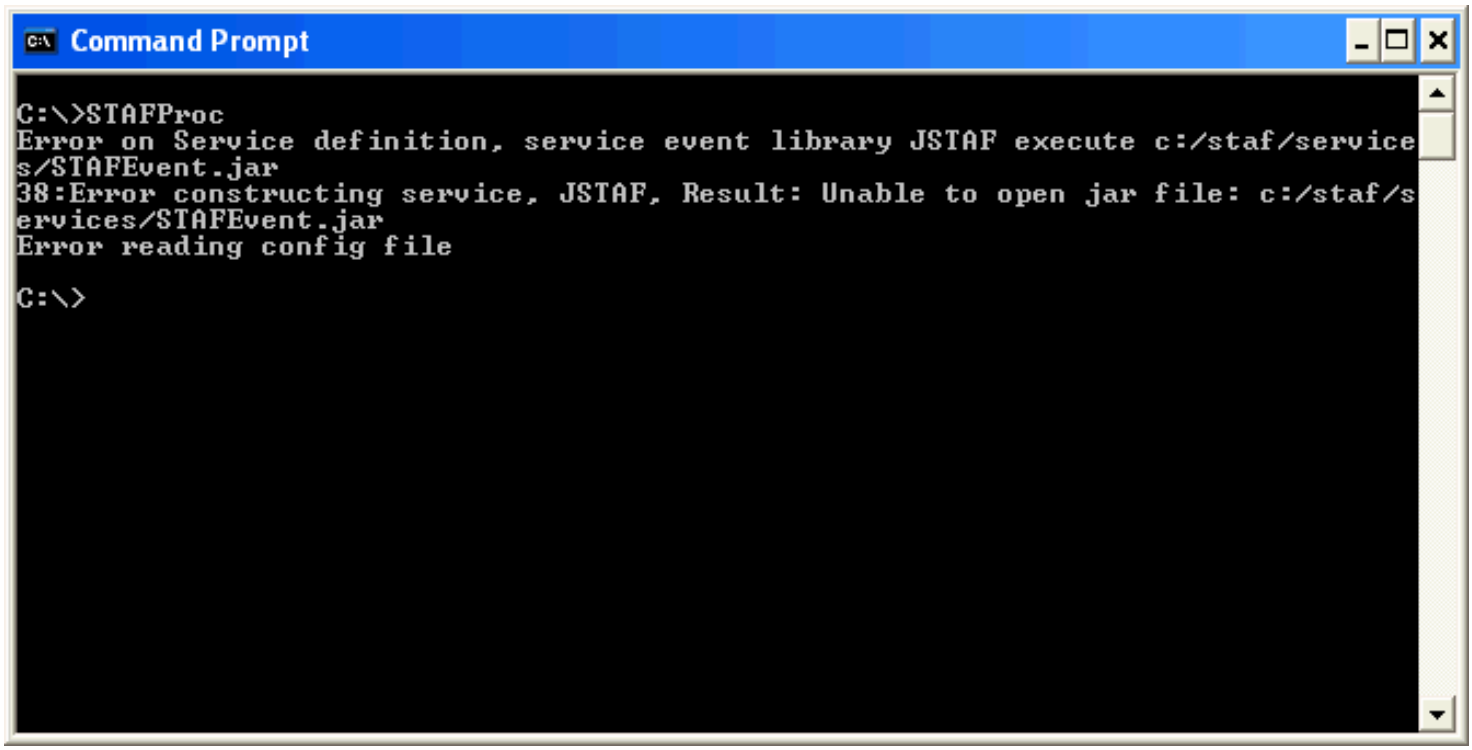
Now let's register a Java service (note that this service is not required to run the Demo, but is included as an example of how to register a Java service). Add the following line to your STAF.cfg file:

```
service Event library JSTAF execute c:/staf/services/STAFEvent.jar
```

Note that the only case-sensitive options in this statement are "JSTAF" and the name of the Jar file. Now restart STAFProc on your machine.

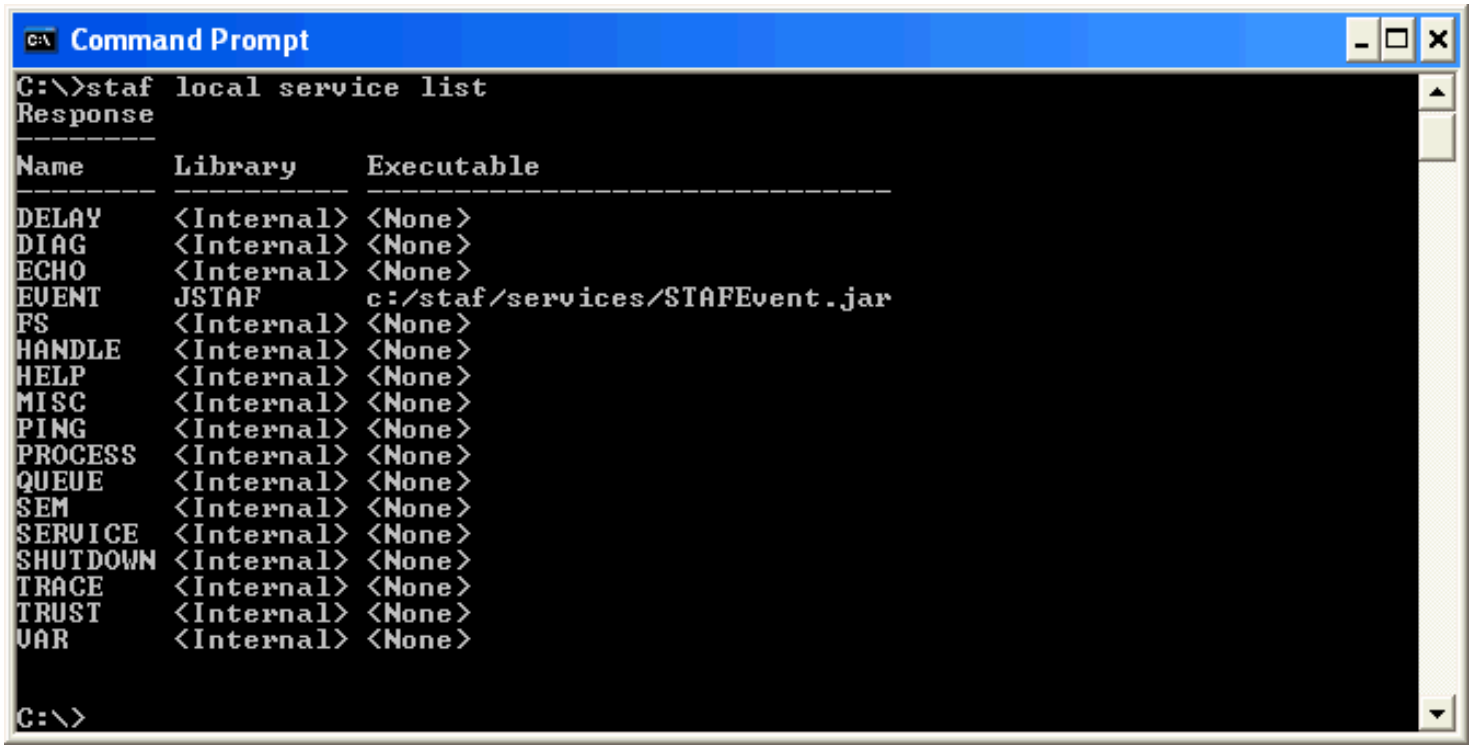
If you are starting the "Start STAF" program from the Windows Start menu, you should see the STAFProc window displayed briefly and then disappear--this means there was a fatal error. To see the error details, start "STAFProc" from a Command Prompt. You should then see the error:

Figure 22.



This demonstrates the fact that the Event Service is not only an External Service, it is also not shipped with the STAF package. There are several Services, including Event, which are available on the staf.sourceforge.net web site. You are required to download the STAFEvent.jar file to your machine (note that the jar file does not need to be in your CLASSPATH). Download the file now to C:\staf\services and restart STAF. Then execute the **`STAF local service list`** command again.

Figure 23.



Now issue a Help request for the Event service, and try to issue some commands to the Event Service. Note that the Event Service User's Guide is available on the "Services" page on the staf.sourceforge.net web site.

In the next section we'll finish the preparation for running the STAF Demo, and then we'll show you how to execute the STAF Demo and examine the code that makes it all work.

8. STAF Demo

8.1. Running the STAF Demo

8.1.1. Configuring the STAF Demo

The STAF Demo is a sample application, written in Java, that demonstrates STAF's capabilities and how to leverage the primary internal and external services in STAF. In particular, it shows the use of the following STAF services:

- Process
- Variable
- Semaphore
- Queue
- Log
- Monitor
- Resource Pool

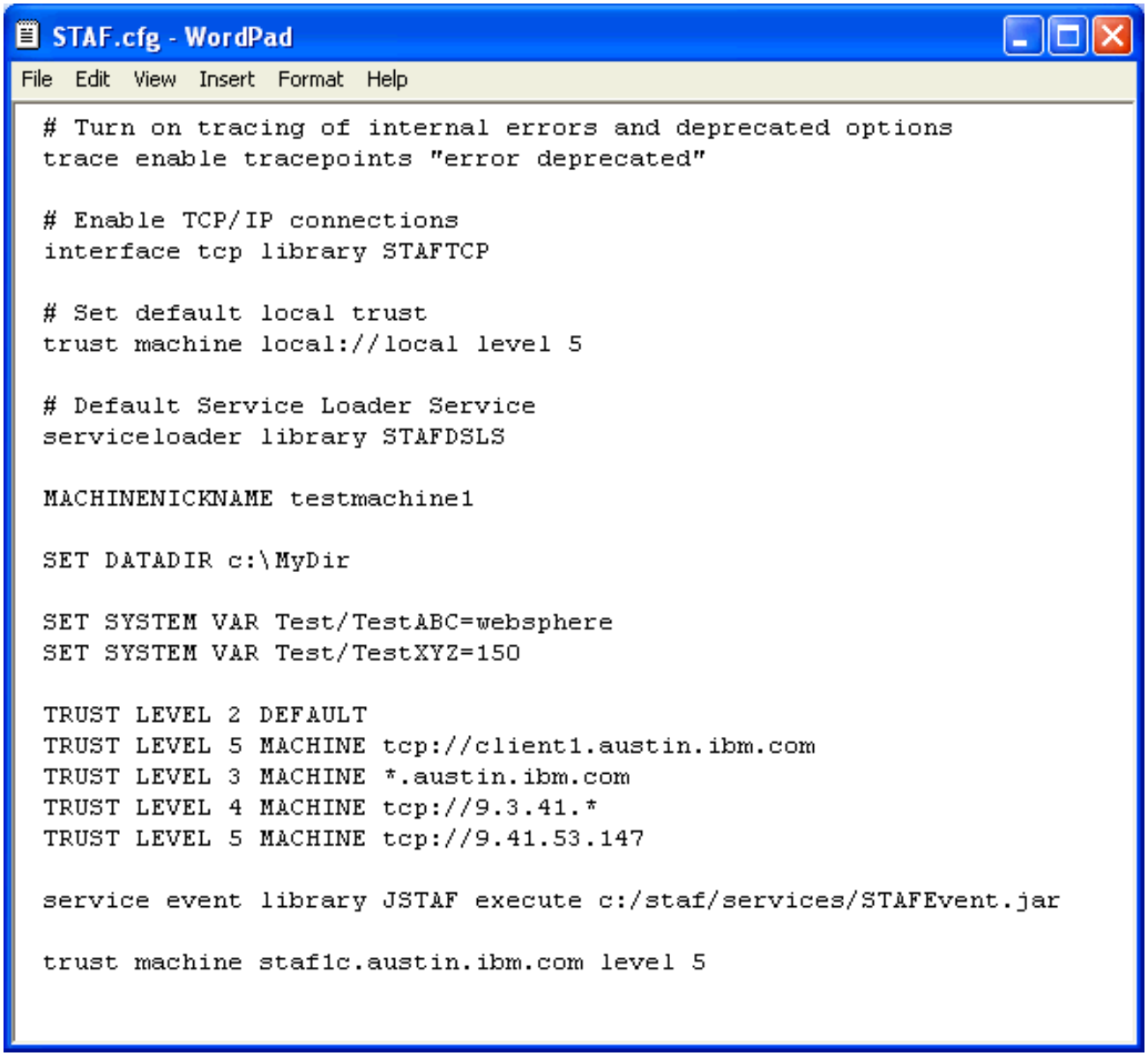
The STAF Demo is shipped with the STAF package. It is located at C:\STAF\samples\demo\STAFDemo.jar.

In order to run the demo, each machine must give the other machine a TRUST level of 5, so you will need to add a TRUST statement to each machine's STAF.cfg file (you can remove the example TRUST entries that were added earlier in this document).

You must also have "C:\STAF\samples\demo\STAFDemo.jar" in your CLASSPATH on your local and remote machines (note, if you selected the default options during the STAF InstallShield installation, this file will already be in your CLASSPATH).

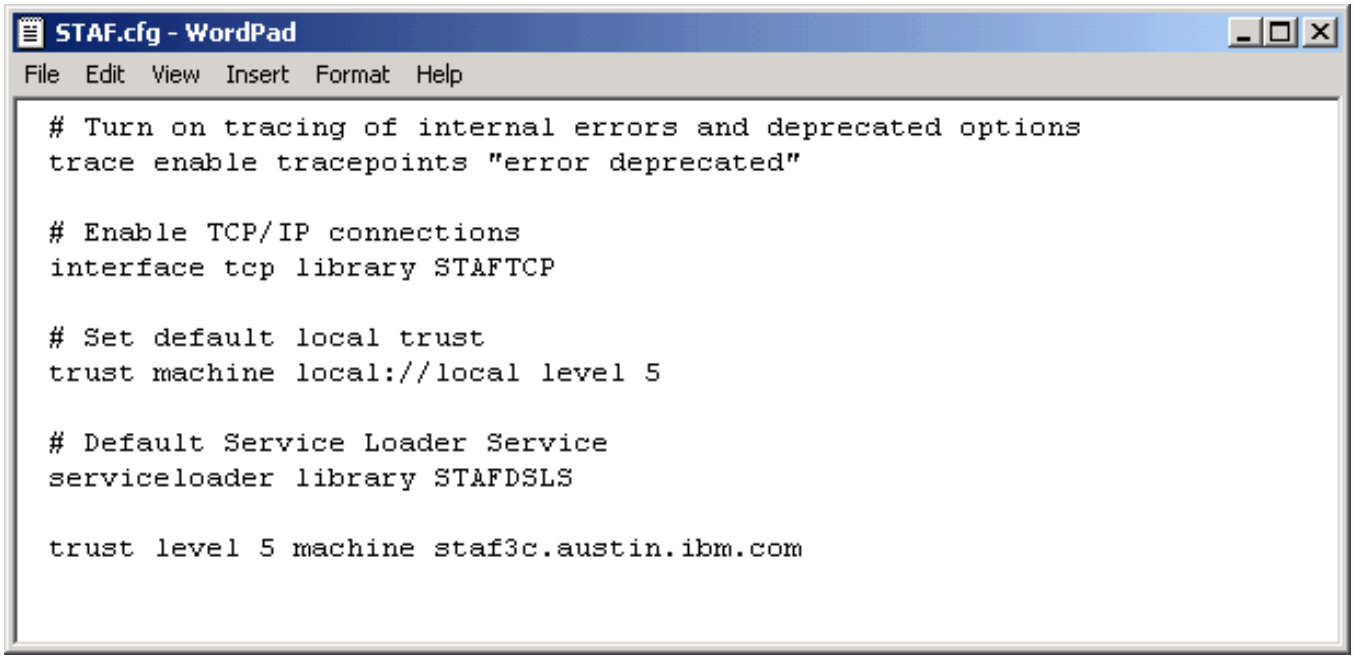
Your local machine's STAF.cfg file should now look similar to:

Figure 24.



Your remote machine's STAF.cfg file should now look similar to:

Figure 25.



The C:\STAF\samples\demo directory also contains the following 2 files, whic are the Java source code for the demo:

- STAFDemoController.java
- STAFProcess.java

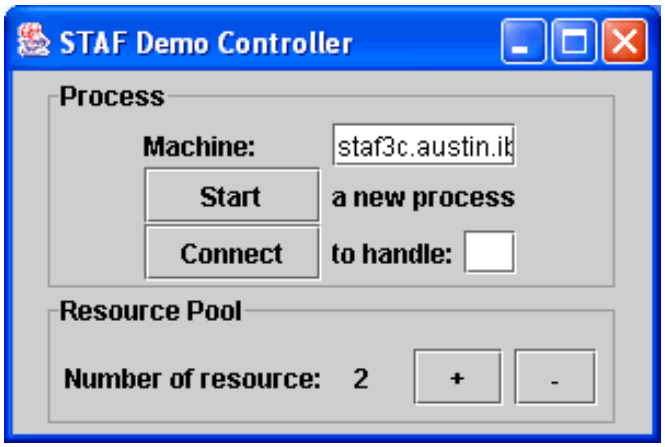
8.1.2. Starting the STAF Demo

At this point you should have the STAF Demo set up on both of your machines, so let's start the demo. First, let's run it locally on one machine. Make sure STAFProc is up and running with your latest STAF.cfg updates. From a command prompt, enter:

```
java STAFDemoController
```

STAFDemoController is the program that drives the demo. You should see the following dialog displayed:

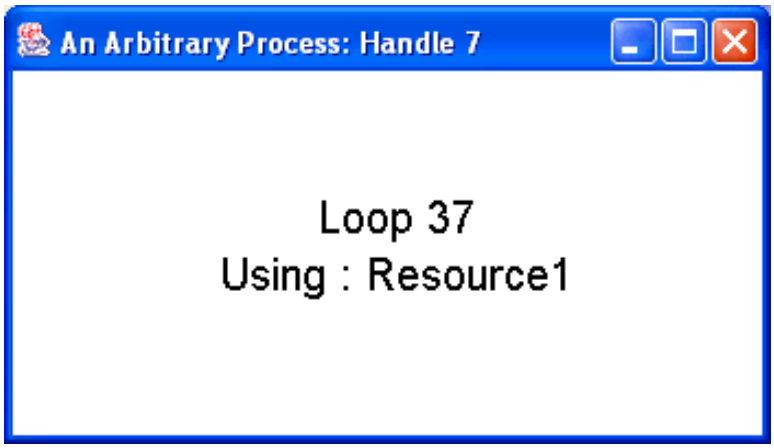
Figure 26.



Now click on the Start button. You should see the following dialogs displayed (note that the exact data, such as timestamps, loop #s, etc., will be different than these images). Note that the windows may overlap so you may need to move the first panel to see the second panel.

Figure 27.

Figure 28.



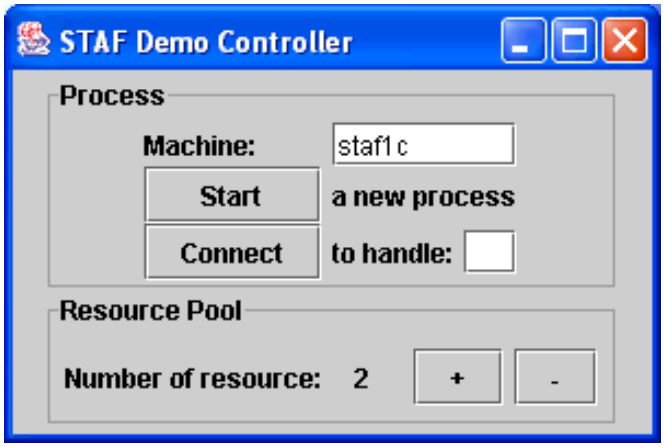
The first dialog ("Machine:....") is the Control window, and the second dialog ("An Arbitrary Process...") is the sample application (STAFProcess). The sample application will loop indefinitely. Note that the titles of the sections in the Control window ("Queue/Semaphore", "Variable", "Monitor", "Log/Variable") indicate which STAF services are being utilized. At this point both the Control window and the application are running on the same local machine.

Click on the Stop button in the Control window (this will cause the "An Arbitrary Process" window to close), and then close the Control window.

Now let's run the demo on a remote system. Make sure that STAFProc is up and running on your remote machine, and verify that it has the STAF Demo correctly set up.

In the STAF Demo Controller window, change "staf3c.austin.ibm.com" to your remote machine (for the tutorial this will be "staf1c"):

Figure 29.

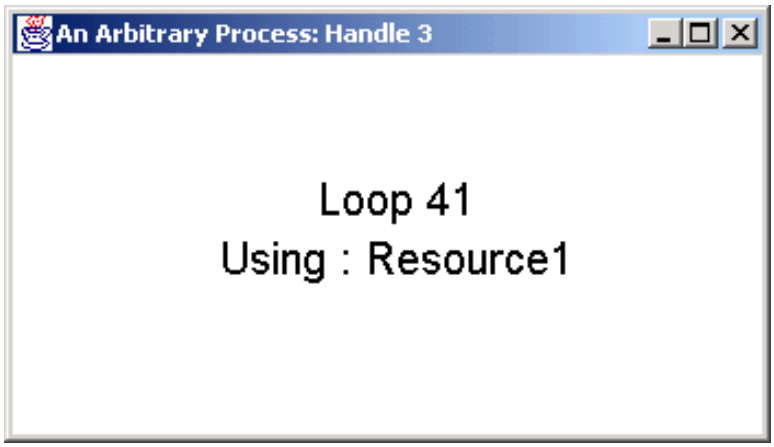


Then click on the Start button. You should still see the Control window on your local machine:

Figure 30.

However, the STAFProcess window should be displayed on your remote machine:

Figure 31.



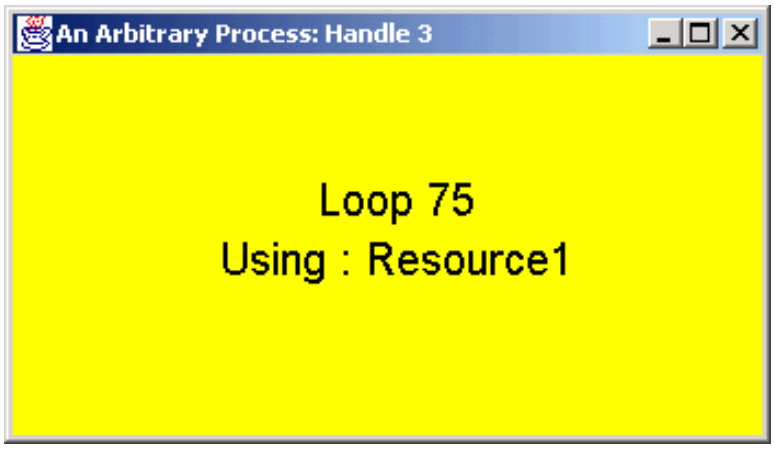
8.1.3. Using the Variable Service to Change the Application's Background Color

Let's try a simple update to the application. In the Control window, notice that the "background color to" combo box is set to "Blue". Select "Yellow" and click on the Set button:

Figure 32.

You should see the update to the sample application on your remote system:

Figure 33.



Note how we are able to change the behavioral characteristics of the application without needing to stop and start the application. At the top of each loop, the application simply checks the value of a STAF variable that defines what its background color should be. This is easily extended to any other type of dynamic information that you would like to be able to change while the application is executing.

8.1.4. Using the Semaphore Service to Control the Application's Execution

Click the "Pause" button in the Control window:

Figure 34.

You should see that the application on the remote system is no longer incrementing its loop counter. Click "Continue" in the Control window and note that the application is continuing its execution. This is accomplished through the semaphore service. The application simply waits on an event semaphore at the top of its loop. This semaphore is normally posted, so the application simply falls on through. When you clicked "Pause", this reset the event semaphore, which caused the application to wait for it to be posted. Clicking "Continue" causes the event semaphore to be posted, which enables the application to continue execution.

8.1.5. Using the Monitor Service to View the Application's Status

The Monitor service allows applications to publish their current status. Look at the "Timestamp:" and "Status:" fields in the "Monitor" group:

Figure 35.

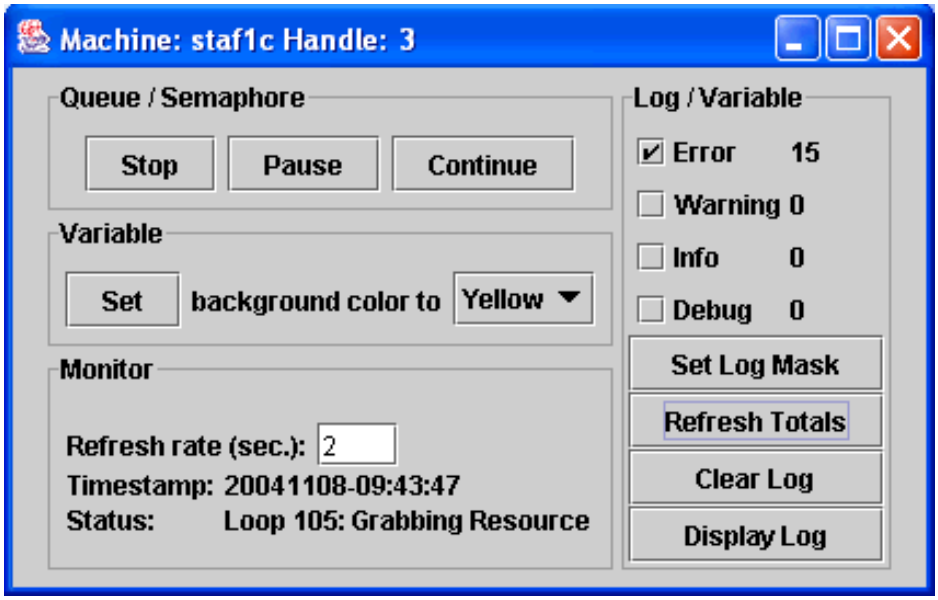
You should see the information updated every few seconds. Note, you can determine where, in execution, the application

is, without needing to actually look at the application display. This is particularly useful when the systems running the applications are not "conveniently" located. In order to do this, the application simply logs occasional messages via the Monitor service.

8.1.6. Using the Log Service to Record Testcase Information

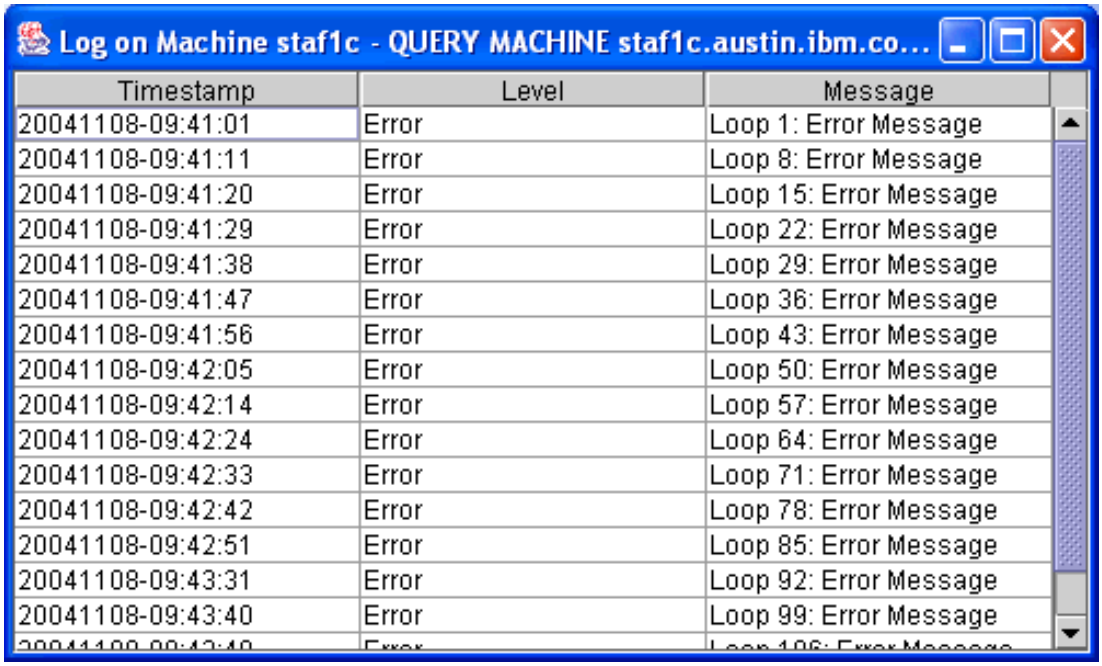
Click "Refresh Totals" in the Control window. You should see that the number of "Error" log messages has increased.

Figure 36.



Click "Display Log" and you should see a new window which contains all the logged error messages.

Figure 37.



Timestamp	Level	Message
20041108-09:41:01	Error	Loop 1: Error Message
20041108-09:41:11	Error	Loop 8: Error Message
20041108-09:41:20	Error	Loop 15: Error Message
20041108-09:41:29	Error	Loop 22: Error Message
20041108-09:41:38	Error	Loop 29: Error Message
20041108-09:41:47	Error	Loop 36: Error Message
20041108-09:41:56	Error	Loop 43: Error Message
20041108-09:42:05	Error	Loop 50: Error Message
20041108-09:42:14	Error	Loop 57: Error Message
20041108-09:42:24	Error	Loop 64: Error Message
20041108-09:42:33	Error	Loop 71: Error Message
20041108-09:42:42	Error	Loop 78: Error Message
20041108-09:42:51	Error	Loop 85: Error Message
20041108-09:43:31	Error	Loop 92: Error Message
20041108-09:43:40	Error	Loop 99: Error Message

Select the "Warning", "Info", and "Debug" checkboxes in the Control window and, then, click on "Set Log Mask".

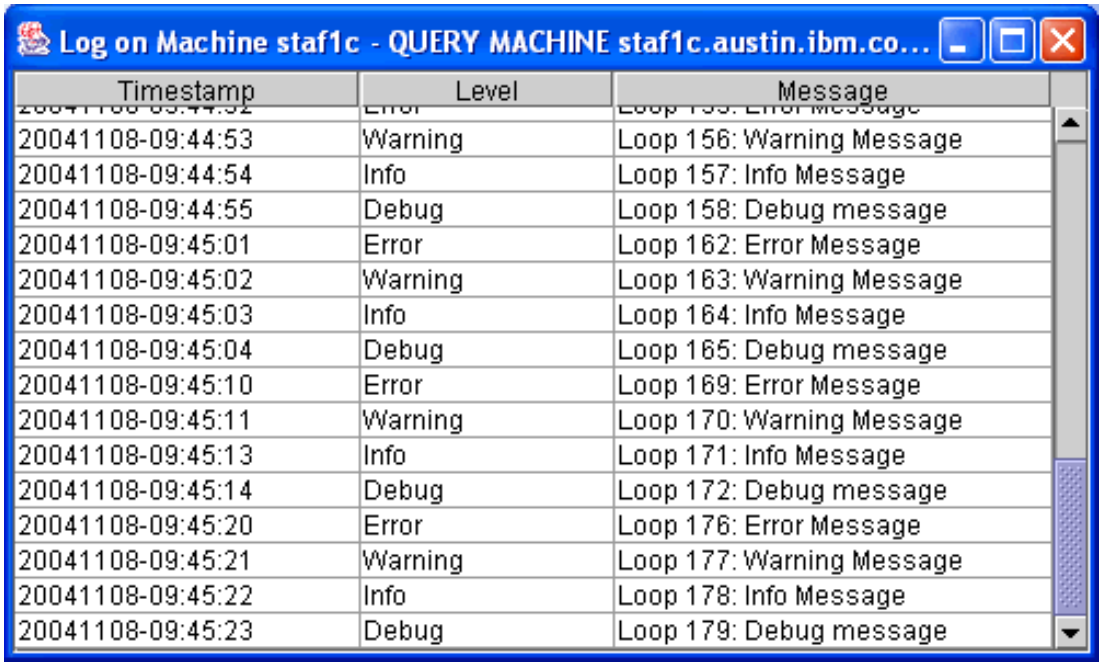
Figure 38.

Now wait for about 30 seconds, and, then, click "Refresh Totals". You should now see that the totals for all the logging levels have increased.

Figure 39.

Click "Display Log" and you should see the other types of log messages at the end of the display.

Figure 40.



This is an example of the dynamic level masking of the STAF Log service. This allows you to define (and change) during runtime the types of messages that are actually being logged into the log file. Note that throughout its execution, the application is actually requesting messages be logged for all 4 types of logs. However, since the log mask was originally set to only "Error", the Log service only wrote "Error" messages to the log. This allows you to have robust logging built into your application, while at the same time being able to dynamically, without changing the application or starting/stopping it, adjust how much information is written to the log.

Click "Clear Log" and you should see all the log totals reset to zero.

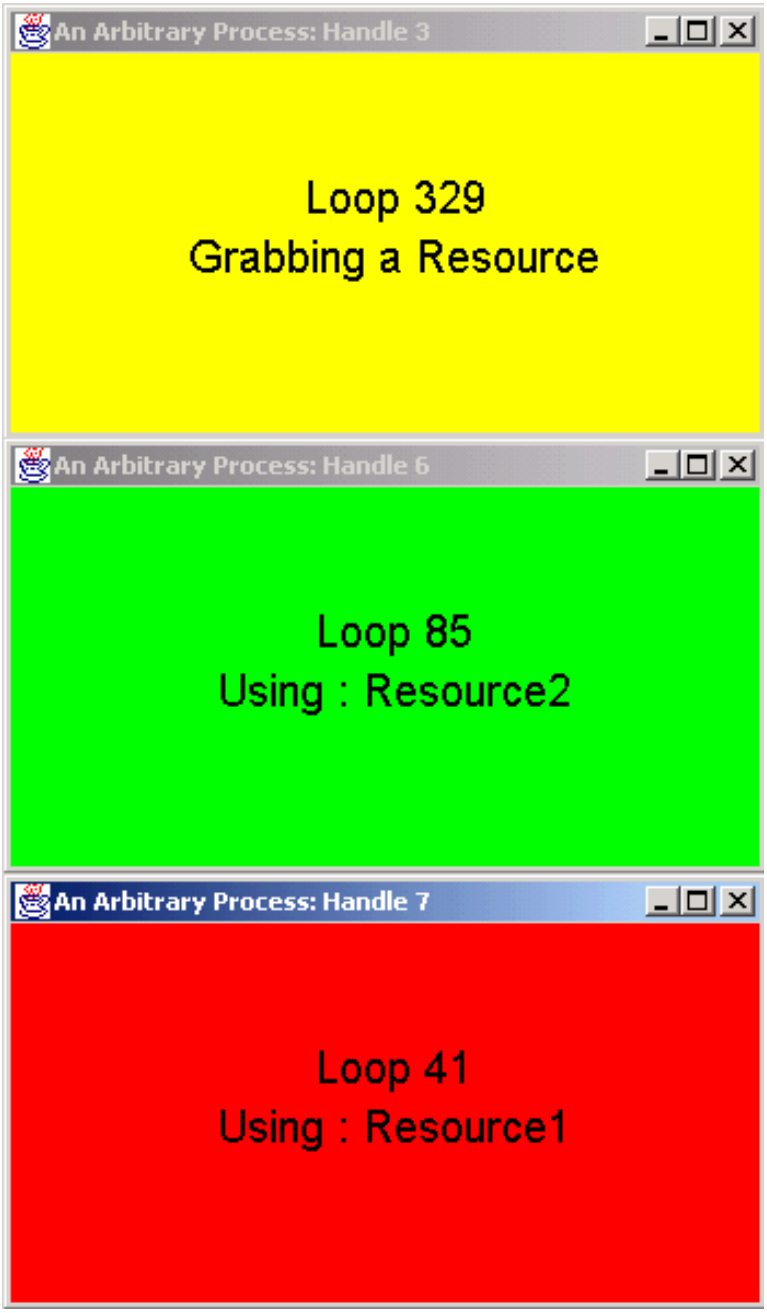
Figure 41.

You have just purged the log, even though the application is still logging data.

8.1.7. Using the Resource Pool Service to Manage Resources

Now, go back to the main control window and start two more STAFProcess applications running on the remote machine. Be sure to change their background colors so that you can distinguish between them. At this point you should have 3 STAFProcess applications running on your remote system:

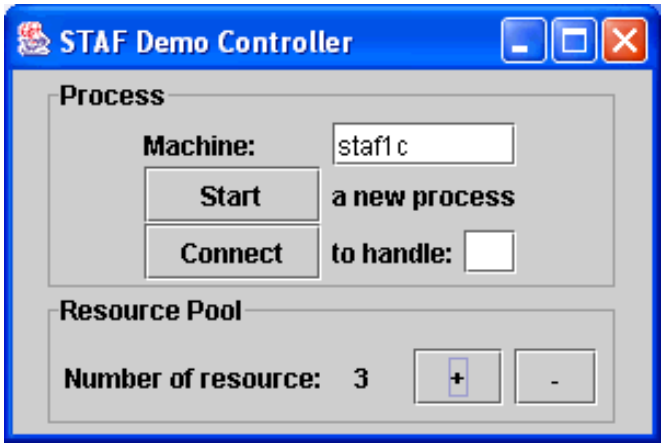
Figure 42.



Since there are initially only two resources available, you should notice that only two out of the three applications will be using a resource at any one time; the other application will be waiting for a resource. This is because each application needs a "resource" in order to continue execution. Periodically, the applications give up their resource and try to acquire a new one. Thus, you should see the applications swapping these two resources amongst themselves.

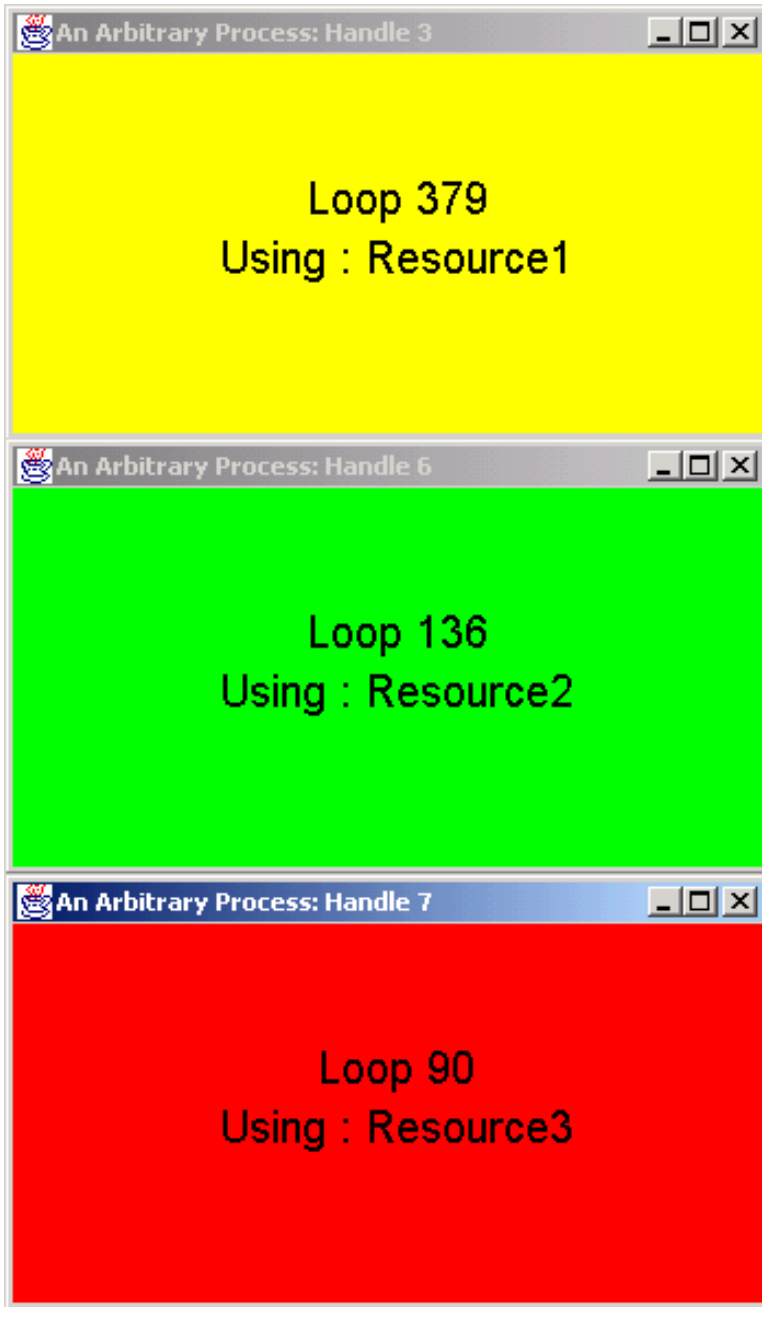
Now, from the main controller window, click the "+" button to the right of "Number of resource".

Figure 43.



This will add another resource for the applications. You should now see all three applications using resources simultaneously.

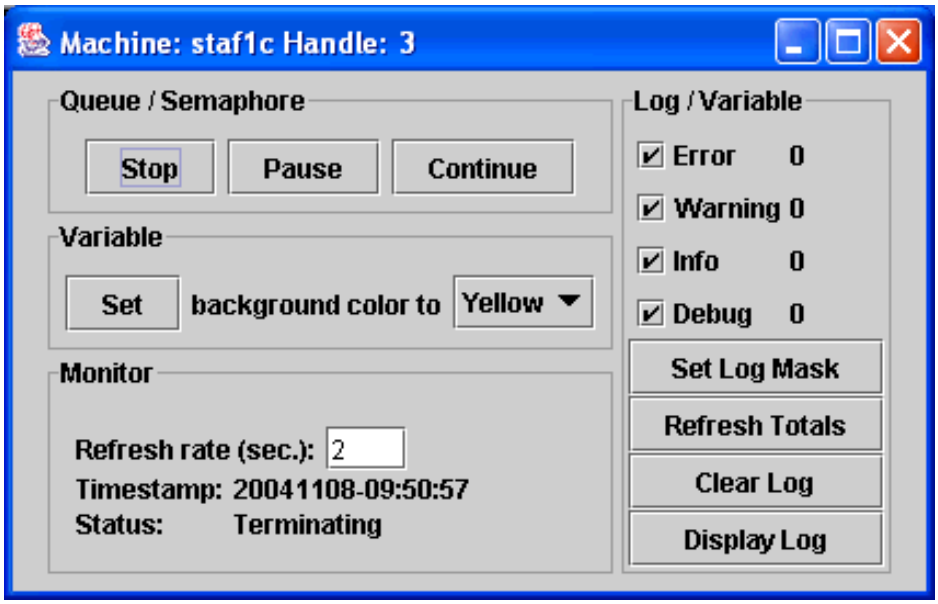
Figure 44.



8.1.8. Using the Queue Service to Send Messages to Testcases

Finally, go to one of the application control windows and click the "Stop" button.

Figure 45.



This will send a message to the application's STAF queue. The application will check this at the top of its loop and terminate gracefully. You should see the application terminate. You should also see the monitor "Status:" display "Terminating". Note, you are still able to retrieve the monitor and log information even after the application has finished.

This is the end of the demo. You can click the "Stop" buttons on the other application controller windows to terminate those applications, and then close the "STAF Demo Controller" window.

So you've now installed STAF on both of your machines, and you are using your local machine to control and monitor STAFProcess applications which are running on your remote machine. Now imagine that you have STAF installed on 100 different machines in a lab (maybe even in different buildings or different coutries), and you're controlling the execution of your application on all 100 machines from your local machine. Hopefully this illustrates the power and capability STAF provides you.

8.2. STAF Demo Code - Leveraging STAF

8.2.1. Registering and Un-registering with STAF

In this section we'll start looking at the actual Java source code for STAFDemoController and STAFProcess, and see exactly how STAF can be leveraged in your test environment. Of course, you don't have to write your applications or services in Java; you could use C, C++, REXX, Perl, Python, TCL, etc.

STAF externalizes four primary classes to Java applications:

STAFHandle	This class handles registering and unregistering with STAF as well as submitting service requests.
STAFException	This class is thrown by STAFHandle when errors are encountered.
STAFResult	This class contains the result of the STAFHandle.submit2() method as well as the STAF error constants.
STAFUtil	This class contains STAF utility functions.

These classes all reside in the com.ibm.staf package. In order to use them in a Java application, you must import the STAF package. Line 26 of STAFDemoController.java shows the import statement.

STAFDemoController.java

```
20:  import java.io.*;
21:  import java.util.*;
22:  import java.awt.*;
23:  import java.awt.event.*;
24:  import javax.swing.*;
25:  import javax.swing.border.*;
26:  import com.ibm.staf.*;
```

The STAFHandle class is used to register, unregister, and submit service requests to STAF. Normally each Java application should create one and only one STAFHandle object. The act of creating this object registers the Java application with STAF using the name passed to the constructor. Let's examine the constructor for STAFDemoController.java:

STAFDemoController.java

```
40:  static STAFHandle handle;
41:  static String stafMachineName;
42:  static String stafMachineNickname;
43:  static int numResources = 2;
44:
45:  // This version of STAFDemo can only communicate with machines
46:  // running STAF V3.x.
47:  static int mainVersion = 3;
48:
49: JPanel stafStartPanel;
50: String targetJavacmd;
51: String targetClasspath;
52:
53: public STAFDemoController()
54: {
55:     super("STAF Demo Controller");
56:
57:     try
```

```
58:     {
59:         handle = new STAFHandle("STAFDemo_Controller");
60:     }
61:     catch(STAFException e)
62:     {
63:         System.out.println("Error registering with STAF, RC: " + e.rc);
64:         System.exit(e.rc);
65:     }
```

Notice on line 40 that a STAFHandle object is defined. Then on line 59, the STAFHandle object is created (passing the name "STAFDemo_Controller"). The "handle" variable will then be used whenever you need to submit requests to STAF. Execution of line 59 effectively registers STAFDemoController with STAF.

The STAFException class is the exception thrown by the STAFHandle class. It contains an rc variable which contains the actual return code from STAF. You may use the standard Throwable method getMessage() to retrieve any extra information provided by STAF. Notice in the constructor for STAFDemoContoller.java, the creation of the STAFHandle is in a try block. On line 61 there is a catch statement for the STAFException class. If an error is encountered while registering with STAF, a STAFException will be thrown. Notice on lines 63 and 64 that you can access the return code by using the rc variable.

Before a Java application exits, it should unregister with STAF by calling the unRegister() method (see line 128 below).

STAFDemoController.java

```
120:  public void windowClosing(WindowEvent event)
121:  {
122:      try
123:      {
124:          handle.submit2(
125:              "local", "QUEUE", "QUEUE HANDLE " + handle.getHandle() +
126:              " TYPE STAF/STAFDemo/Stop MESSAGE " +
127:              STAFUtil.wrapData(""));
128:          handle.unRegister();
129:      }
130:      catch(STAFException e)
131:      {
132:          System.out.println("Error unregistering with STAF, RC: " +
133:              e.rc);
134:      }
135:
136:      dispose();
137:      System.exit(0);
138:  }
```

8.2.2. Submitting Requests to STAF

Service requests may be submitted by one of two methods:

- The submit() method works in the traditional Java fashion, in that it throws an exception (a STAFException in particular) if it encounters an error, and it returns a result string on success.
- The submit2() method returns a STAFResult object in all cases. This object contains the real STAF return code as well as the result string. It is typically used in places where you wish to avoid catching exceptions when using STAF.

Let's take another look at the constructor for STAFDemoController.java:

STAFDemoController.java

```
53: public STAFDemoController()
54: {
55:     super("STAF Demo Controller");
56:
57:     try
58:     {
59:         handle = new STAFHandle("STAFDemo_Controller");
60:     }
61:     catch(STAFException e)
62:     {
63:         System.out.println("Error registering with STAF, RC: " + e.rc);
64:         System.exit(e.rc);
65:     }
66:
67:     STAFResult stafResult = handle.submit2(
68:         "local", "VAR", "RESOLVE STRING {STAF/Config/Machine}");
69:
70:     if (stafResult.rc != 0)
71:         System.out.println("Error getting local machine name RC: "+
72:             stafResult.rc + " Result: " + stafResult.result);
73:
74:     stafMachineName = stafResult.result;
```

On line 67 there is a call to submit2(). This method takes 3 parameters:

- The first parameter is where the STAF request is to be executed. In this case, it will be executed locally.
- The second parameter is the STAF Service where the request should be routed. In this case, it is the "var" (Variable) service.
- The third parameter is the actual request to be sent to the STAF Service. In this case the request is to resolve the string "{STAF/Config/Machine}".

This call to submit2() should return the local machine name. Notice on line 67 that the result of the submit2() call is a STAFResult object. On line 70, the rc variable of this object is examined to determine if the request was successful. On line 74, the result variable of the STAFResult object is accessed to get the String result (the local machine name).

8.2.3. Using the Process Service

During the Demo, when you click on the "Start" button to start the application, the Controller uses the Process service to start the application:

STAFDemoController.java

```
399:  stafResult = handle.submit2(
400:       machineName, "PROCESS", "START COMMAND " +
401:       targetJavacmd + " " + "PARMS STAFProcess " +
402:       "VAR STAFDemo/ResourcePoolMachine=" + stafMachineName +
403:       " VAR STAF/Service/Log/Mask=Error" +
404:       " ENV CLASSPATH=" + targetClasspath + " NOTIFY ONEND");
405:
406:  if (stafResult.rc != 0)
407:  {
408:       String errMsg = "Error starting process.  RC:" +
409:           stafResult.rc + " Result:" + stafResult.result;
```

Starting on line 399, there is a call to submit2 which starts the application on the remote system.

8.2.4. Using the Variable Service

While executing the STAF Demo, you modified the background color of the sample application by selecting a new color in the Control window. Let's look at how this code works.

STAFDemoController.java

```
719:  fSetBackground.addActionListener(new ActionListener() {
720:      public void actionPerformed(ActionEvent event)
721:      {
722:          STAFResult stafResult =
723:              STAFDemoController.handle.submit2(
724:                  fMachine, "VAR", "SET HANDLE " + fHandle +
725:                   " VAR STAFDemo/BackgroundColor=" +
726:                   (String)fColorList.getSelectedItem());
727:
728:          if (stafResult.rc != 0)
729:              System.out.println("Error setting background color RC: " +
730:                  stafResult.rc + " Result: " + stafResult.result);
731:      }
732:  });
```

When you click on the "Set" button to change the background color, the actionPerformed method at line 720 in STAFDemoController is called.

On line 723, there is a call to submit2:

- The first parameter is the machine where the application is executing.
- The second parameter is the service name, in this case the Variable service.
- The third parameter is the service request. In this case we first pass the handle, and then set the STAFDemo/BackgroundColor variable to the color selected by the user. Note that we are setting this variable specifically for this application's handle.

As already stated, the application is running in a loop:

STAFProcess.java

```
134:  public void run()
135:  {

172:      while (true)
173:      {

335:          // sleep for 1 second before looping again
336:          handle.submit2(machine, "DELAY", "DELAY 1000");
337:          counter++;
338:      }
```

Now let's look at how the application processes the changes made to the STAFDemo/BackgroundColor variable.

STAFProcess.java

```
154:  String background_color_var = new String(
155:      "resolve string {STAFDemo/BackgroundColor}");

205:  // get the background color, save the old one
206:  previous_color = (color == null ? null : new String(color.result));
207:  color = handle.submit2(machine, "VAR", background_color_var);
208:
209:  if (color != null && color.rc == STAFResult.Ok)
210:  {
211:      // if color changed, log an informational message
212:      if (previous_color != null &&
213:          !previous_color.equals(color.result))
214:      {
215:          // use monitor as a checkpoint
216:          monitor = handle.submit2(
217:              machine, "MONITOR", "LOG MESSAGE \"Loop " +
218:              String.valueOf(counter) + ": Changing Color\"");
219:      }
220:
221:      // set background according to color
222:      frame.getContentPane().setBackground(
223:          getColorFromString(color.result));
224:  }
225:  else
226:  {
227:      // set background to default (white)
228:      frame.getContentPane().setBackground(Color.white);
229:  }
```

On line 207, the application calls submit2 to retrieve the value of the color variable from the Variable service. The variable background_color_var is defined on line 154.

Notice that by using STAF's Variable service, we are able to change an operational parameter in the application without having to stop and restart it.

8.2.5. Using the Semaphore and Queue Services

In the Demo Control window, if you click on the "Pause" button, the application will Pause. If you then click on the "Continue" button, the application will resume. Let's take a look at how this code is implemented.

STAFDemoController.java

```
691:  fPause.addActionListener(new ActionListener() {
692:      public void actionPerformed(ActionEvent event)
693:      {
694:          STAFResult stafResult =
695:              STAFDemoController.handle.submit2(
696:              fMachine, "SEM", "RESET EVENT STAFDemo/Handle/" +
697:              fHandle + "/Continue");
698:
699:          if (stafResult.rc != 0)
700:              System.out.println("Error pausing process RC: " +
701:                  stafResult.rc + " Result: " + stafResult.result);
702:      }
703:  });
```

On line 695 there is a call to the Semaphore service on the machine where the application is running. This call will reset the event semaphore which is uniquely identified by the application's handle.

Now let's look at how the application uses this semaphore":

STAFProcess.java

```
156:  String continue_semaphore = new String(
157:      "wait event STAFDemo/Handle/"+h+"/Continue");

196:  // block if semaphore is reset, fall through if posted (or
197:  // if error!!!)
198:  semaphore = handle.submit2(machine, "SEM", continue_semaphore);
```

In STAFProcess, on line 198 there is a call to the local Semaphore service. The variable continue_semaphore is defined on line 156.

This call to submit2 will cause application to wait for the event semaphore uniquely identified by its handle. Thus, whenever STAFDemoController resets the event semaphore, the application will then wait for the event semaphore, effectively pausing the application's execution.

Now let's examine the code that resumes the application's execution:

STAFDemoController.java

```
705:  fContinue.addActionListener(new ActionListener() {
706:      public void actionPerformed(ActionEvent event)
707:      {
708:          STAFResult stafResult =
709:              STAFDemoController.handle.submit2(
710:                  fMachine, "SEM", "POST EVENT STAFDemo/Handle/" +
```

```
711:                  fHandle + "/Continue");
712:
713:          if (stafResult.rc != 0)
714:              System.out.println("Error continuing process RC: " +
715:                  stafResult.rc + " Result: " + stafResult.result);
716:      }
717:  });
```

When you click on the "Continue" button, line 708 in STAFDemoController will again submit a request to the Semaphore service on the machine where application is running. This time it will post the event semaphore, so that application (still blocked on line 198) will continue execution. Note that the Semaphore service provides mutex as well as event semaphores.

While continuing and pausing the application utilizes the Semaphore service, stopping the application utilizes the Queue Service. The Queue service is how processes communicate with each other. In this case, the Controller sends a message to the application's queue. This message type indicates to the application that it should terminate.

STAFDemoController.java

```
676:  fStop.addActionListener(new ActionListener() {
677:      public void actionPerformed(ActionEvent event)
678:      {
679:          STAFResult stafResult =
680:              STAFDemoController.handle.submit2(
681:              fMachine, "QUEUE", "QUEUE HANDLE " + fHandle +
682:              " TYPE STAF/STAFDemo/Stop MESSAGE " +
683:              STAFUtil.wrapData("));
684:
685:          if (stafResult.rc != 0)
686:              System.out.println("Error queueing message RC: " +
687:                  stafResult.rc + " Result: " + stafResult.result);
688:      }
689:  });
```

When you click on the "Stop" button, line 679 of STAFDemoController will submit a request to the Queue service on the machine where the application is executing. This will result in a message with type "STAF/STAFDemo/Stop" being placed on the queue for the specified handle. Now let's look at how the application processes this message.

STAFProcess.java

```
159:  String mesg_queue = new String("GET TYPE STAF/STAFDemo/Stop");

182:  // check queue for stop message
183:  stop = handle.submit2(machine, "queue", mesg_queue);
184:
185:  if (stop != null && stop.rc == STAFResult.Ok)
186:  {
187:      // break from the loop
188:      break;
189:  }
```

In STAFProcess' infinite while loop, on line 183 there is a call to the local Queue service. The variable mesg_queue is

defined on line 159.

This will check the Queue for the application's handle to determine if there is a message with type "STAF/STAFDemo/Stop" on its queue. When the STAFDemoController places this message on the application's queue, the stop variable will not be null and its return code will be STAFResult.Ok, so the application will then break out of its infinite while loop, thus terminating the application.

8.2.6. Using the Log and Monitor Services

The STAF Demo uses both the Log and Monitor services. Let's look at an example of where the application writes to the Log and Monitor services.

STAFProcess.java

```
276:  // randomly log an error, warning, debug or information message
277:  switch (counter % 7)
278:  {
279:      // error
280:      case 1:
281:          // use monitor as a checkpoint
282:          monitor = handle.submit2(
283:              machine, "MONITOR", "LOG MESSAGE \"Loop " +
284:              String.valueOf(counter)+": Logging Error\"");
285:
286:          // do the logging
287:          handle.submit2(
288:              machine, "LOG", mesg_log + "LEVEL ERROR " +
289:              "MESSAGE \"Loop " + String.valueOf(counter) +
290:              ": Error Message\"");
291:          break;
292:      // warning
293:      case 2:
294:          // use monitor as a checkpoint
295:          monitor = handle.submit2(
296:              machine, "MONITOR", "LOG MESSAGE \"Loop " +
297:              String.valueOf(counter)+": Logging Warning\"");
298:
299:          // do the logging
300:          handle.submit2(
301:              machine, "LOG", mesg_log + "LEVEL warning " +
302:              "MESSAGE \"Loop " + String.valueOf(counter) +
303:              ": Warning Message\"");
304:          break;
```

Note that on line 282, a request is submitted to the Monitor service to write an error message. On line 287, a request is submitted to the Log service to log an error message. "case 2" starting on line 293 is similar, except that a warning is written instead of an error.

Now let's examine how the STAFDemoController retrieves the Monitor information.

STAFDemoController.java

```
994:   STAFResult stafResult =
995:       STAFDemoController.handle.submit2(
```

```
996:       fMachine, "MONITOR", "QUERY MACHINE " +
997:       fMachineNickname + " HANDLE " + fHandle);
998:
999:   if (stafResult.rc != 0)
1000:  {
1001:      fDateField.setText("Unknown");
1002:      fStatusField.setText("Unknown");
1003:  }
1004:  else
1005:  {
1006:      if (STAFMarshallingContext.isMarshalledData(
1007:          stafResult.result))
1008:      {
1009:          // STAF 3.x - Unmarshall the result
1010:
1011:          STAFMarshallingContext outputContext =
1012:              STAFMarshallingContext.unmarshall(
1013:                  stafResult.result);
1014:          Map resultsMap =
1015:              (Map)outputContext.getRootObject();
1016:          fDateField.setText(
1017:              (String)resultsMap.get("timestamp"));
1018:          fStatusField.setText(
1019:              (String)resultsMap.get("message"));
1020:      }
1021:      else
1022:      {
1023:          // STAF V2.x results are not marshalled
1024:
1025:          int firstSpace = stafResult.result.indexOf(' ');
1026:          String dateText =
1027:              stafResult.result.substring(0, firstSpace);
1028:          String messageText =
1029:              stafResult.result.substring(firstSpace + 1);
1030:          fDateField.setText(dateText);
1031:          fStatusField.setText(messageText);
1032:      }
1033:  }
```

STAFDemoController has a separate thread that periodically queries the application's Monitor. This call is on line 994. The Timestamp and Status information is updated based on the request result.

Note that this example also demonstrates the differences between multi-value results on STAF 2.x and 3.x. In STAF 2.x, the result from the MONITOR QUERY was a string, with fields separated by a semi-colon, that you would need to parse though to get the fields. In STAF 3.x, the result is now a marshalled data structure, and there are STAF Java APIs that can be used to unmarshall the structure so that you directly access the fields.

Now let's examine how the STAFDemoController retrieves the Log information.

STAFDemoController.java

```
826:  String request = "QUERY MACHINE " + fMachineNickname +
```

```
827:      " HANDLE " + fHandle + " LOGNAME STAFDemo";
828:
829:  STAFResult stafResult = STAFDemoController.handle.submit2(
830:      fMachine, "LOG", request);
831:
832:  if (stafResult.rc == 0)
833:  {
834:      // Unmarshall the output from the request and create a
835:      // outputList containing the results (timestamp, level,
836:      // and message)
837:
838:      try
839:      {
840:          STAFMarshallingContext outputContext =
841:              STAFMarshallingContext.unmarshall(
842:                  stafResult.result);
843:
844:          outputList = (java.util.List)outputContext.
845:              getRootObject();
846:      }
```

```
869:  // Create a vector (logLines) from the outputList
870:
871:  Iterator iter = outputList.iterator();
872:  Vector logLines = new Vector();
873:  int i = 0;
874:
875:  try
876:  {
877:      while (iter.hasNext())
878:      {
879:          i++;
880:          Map logRecord = (Map)iter.next();
881:          Vector thisLogData = new Vector();
882:
883:          thisLogData.add((String)logRecord.get("timestamp"));
884:          thisLogData.add((String)logRecord.get("level"));
885:          thisLogData.add((String)logRecord.get("message"));
886:          logLines.add(thisLogData);
887:      }
888:  }
```

When you click on the "Display Log" button, on line 829 of STAFDemoController, a request will be submitted to the Log service to query the log information. The result is then displayed in the table.

8.2.7. Using the Resource Pool Service

Let's investigate how the Demo Controller creates the Resource Pool used in the Demo.

STAFDemoController.java

```
92:   stafResult = handle.submit2(
93:       "local", "RESPOOL", "CREATE POOL STAFDemo DESCRIPTION \"" +
94:       "STAF Demo Resource Pool\"");
95:
96:   if (stafResult.rc != 0)
97:       System.out.println("Error creating STAFDemo resource pool RC: " +
98:           stafResult.rc + " Result: " + stafResult.result);
99:
100:  for (int i = 1; i <= numResources; ++i)
101:  {
102:      stafResult = handle.submit2(
103:          "local", "RESPOOL", "ADD POOL " + "STAFDemo ENTRY Resource" +
104:          String.valueOf(i));
105:
106:      if (stafResult.rc != 0)
107:          System.out.println("Error adding resource to  STAFDemo RC: "+
108:              stafResult.rc + " Result: " + stafResult.result);
109:  }
```

In STAFDemoController, on line 92 a Resource Pool titled "STAFDemo" is created. Then, on line 102, several resource entries are added to the pool (the number of these entries can be changed on the main STAF Demo Controller panel).

Now let's examine how the application requests one of the resources:

STAFProcess.java

```
158:  String items_respool = new String("REQUEST POOL STAFDemo RANDOM");

257:  // block until resource is available
258:  resource = handle.submit2(
259:      "{STAFDemo/ResourcePoolMachine}", "RESPOOL",
260:      items_respool);
```

On line 258 of STAFProcess, a request is submitted to the ResPool service to request a resouce. The variable items_respool is defined on line 158. The application will block until one of the resources becomes available.

The Resource Pool Service can be used for a wide range of functions: controlling access to Userids, printer allocation management, control of software licence distribution, etc.

9. Glossary

external services	Services for which the executable code for the service resides outside of STAFProc, and which must be registered in the STAF configuration file. Some external services, such as LOG, are provided with STAF. Others are available from the STAF web site and must be downloaded and installed.
handle	A unique identifier which is used when submitting requests to STAF.
internal services	Services for which the executable code resides within STAFProc.
machine name	A unique string (typically the machine's TCP/IP host name) which identifies different systems in the STAF Environment.
marshall	To take a data structure and convert it into a string-based representation.
process	A process is an object which can be executed on a test machine. Examples of processes are: executables, shell scripts, etc.
request	A string, sent to a service, which describes the operation the service is to perform.
services	Reusable components that provide all the capability in STAF. Each STAF service provides a specific set of functionality and defines a set of requests that it will accept.
STAF	An automation framework designed around the idea of reusable components
STAF command	A STAF Command consists of a machine, service, and request. The request is sent to the machine, the service on the machine processes the request, and returns a return code and a result, if any.
STAF configuration file	A file which contains configuration statements for STAF.
STAF environment	The collection of machines on which you have installed STAF.
STAFProc	The daemon process which runs on each STAF system.
unmarshall	To convert a string-based representation back into a data structure.
workload	A set of processes running on a set of machines.