

Getting Started With STAF

Version 1.1.0

Document Owner: David Bender
11400 Burnet Road
Austin, TX 78758
Telephone: (512)838-1268

16 July 2002

TOPICS

[OVERVIEW](#)

[STAF RESOURCES](#)

[INSTALLING STAF](#)

[BASIC STAF CONCEPTS](#)

[STAF SERVICES](#)

[STAF COMMANDS](#)

[CONFIGURING STAF](#)

[USING THE HELP SERVICE](#)

[REGISTERING SERVICES](#)

[STAF Demo](#)

[Running the Demo](#)

[Registering and Un-registering with STAF](#)

[STAF DEMO - Using the Process Service](#)

[Submitting Requests to STAF](#)

[Using the Variable Service](#)

[Using the Semaphore and Queue Services](#)

[Using the Log and Monitor Services](#)

[Using the Resource Pool Service](#)

[GLOSSARY](#)

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 V2.4. Older releases of STAF may not have the same functionality that is described in this document.

STAF RESOURCES

The official SourceForge STAF web site is at: <https://sourceforge.net/projects/staf/> 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.




STAF User's Guide

An extremely useful document on the STAF web site is the STAF User's Guide. It contains detailed information on how to set up STAF and use STAF commands and Services. While this Tutorial covers many of the same topics, its focus is on getting you familiar with using STAF; for more in-depth information (syntax, error codes) you can refer to the STAF User's Guide.

Mailing Lists

There are 3 Mailing Lists on the SourceForge STAF web site. You can access them by clicking "Mailing Lists" from the SourceForge STAF web site:

Choose a list to browse, search, and post messages.

-  [staf-devel Archives](#) (go to [Subscribe/Unsubscribe/Preferences](#))
For developer use only
-  [staf-news Archives](#) (go to [Subscribe/Unsubscribe/Preferences](#))
Low traffic, read-only list for news and announcements
-  [staf-users Archives](#) (go to [Subscribe/Unsubscribe/Preferences](#))
Questions, suggestions, support, and general discussion of STAF

IBM Intranet News Groups

The IBM Intranet STAF news groups allow IBM Employees to participate in discussions regarding STAF and to obtain assistance both from the 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 current STAF news groups are shown below:



INSTALLING STAF

The STAF User's Guide contains instructions on how to install STAF, but there are a few caveats that you should be aware of.

- When you are replacing an existing version of STAF, it is recommended that you first uninstall the existing version of STAF before proceeding with the new installation. The STAF User's Guide contains the uninstall procedures. It is also recommended that you always save a backup copy of your existing STAF.cfg file (this file, and its usage, are described in the "CONFIGURING STAF" Topic).
- When installing STAF on Unix platforms, you are required to run the STAFInst script (it is not sufficient to simply untar the compressed STAF image).
- On Windows, make sure that your System PATH includes "c:\staf\bin" (assuming that you installed STAF in c:\staf). Also make sure that "c:\staf\bin" and "c:\staf\bin\JSTAF.zip" are in your System CLASSPATH.

Note: 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.

Note: 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.

BASIC STAF CONCEPTS

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.



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) and defines a set of requests (see below) that it will accept.

STAF services will be discussed in more detail in the [STAF SERVICES](#) Topic.

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.

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 of the machine.

STAF Handles

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

It is this combination of machine 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.

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 global variable pool that is common to all the processes on a STAF machine. Additionally, each handle has its own variable pool.

By default, the values of variables in a process' variable pool override the values of variables in the global variable pool. However, the process may override this behavior when asking for the value of a variable.

STAF Security

STAF manages security at the machine name level, as opposed to using individual userids.

Users assign a numeric *trust* level to specific machines names. Other machines in the STAF Environment receive a default user-configurable trust level.

Each STAF service defines the trust levels that are required for each of its requests.

STAF Queues

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

Applications receive messages sent from other handles on their queue.

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 below in the [STAF COMMANDS](#) Topic).

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.

STAF SERVICES

Internal STAF Services

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

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 in section [REGISTERING SERVICES](#).

Note that you may want to install and register some external STAF services (e.g. 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.

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.

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 serviceloaders. 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 the Log, Monitor, and ResPool services.

Custom STAF Services

Note that you can also write your own custom services that can be plugged into STAF.

STAF Services provided:

The following are some of the services that STAF provides:

INTERNAL STAF SERVICES:

FILE SYSTEM	Allows you to get and copy files across the network	<i>Internal ("FS")</i>
HANDLE	Provides information about existing STAF handles	<i>Internal ("HANDLE")</i>
HELP	Provides Help on STAF error codes	<i>Internal ("HELP")</i>
PING	Provides a simple is-alive message	<i>Internal ("PING")</i>
PROCESS	Allows you to start, stop, and query processes	<i>Internal ("PROCESS")</i>
QUEUE	Provides a network-enabled IPC mechanism for STAF Programs	<i>Internal ("QUEUE")</i>
SEMAPHORE	Provides network-enabled named event and mutex semaphores	<i>Internal ("SEM")</i>
SERVICE	Allows you to list services available on a machine and to examine the Requests that have been submitted on a machine	<i>Internal ("SERVICE")</i>
SHUTDOWN	Provides a means to shutdown STAF and register for shutdown notifications	<i>Internal ("SHUTDOWN")</i>
VARIABLE	Provides a method for maintaining configuration and runtime data (variables)	<i>Internal ("VAR")</i>

EXTERNAL STAF SERVICES:

EVENT	Provides a publish/subscribe notification system	<i>External (Java)</i>
LOG	Provides a full-featured logging facility	<i>External (C++)</i>
RESOURCE POOL	Allows you to manage exclusive access to pools of elements, e.g. VM UserIDs or Software Licenses	<i>External (C++)</i>
MONITOR	Allows a testcase to publish its current running execution status for others to read	<i>External (C++)</i>

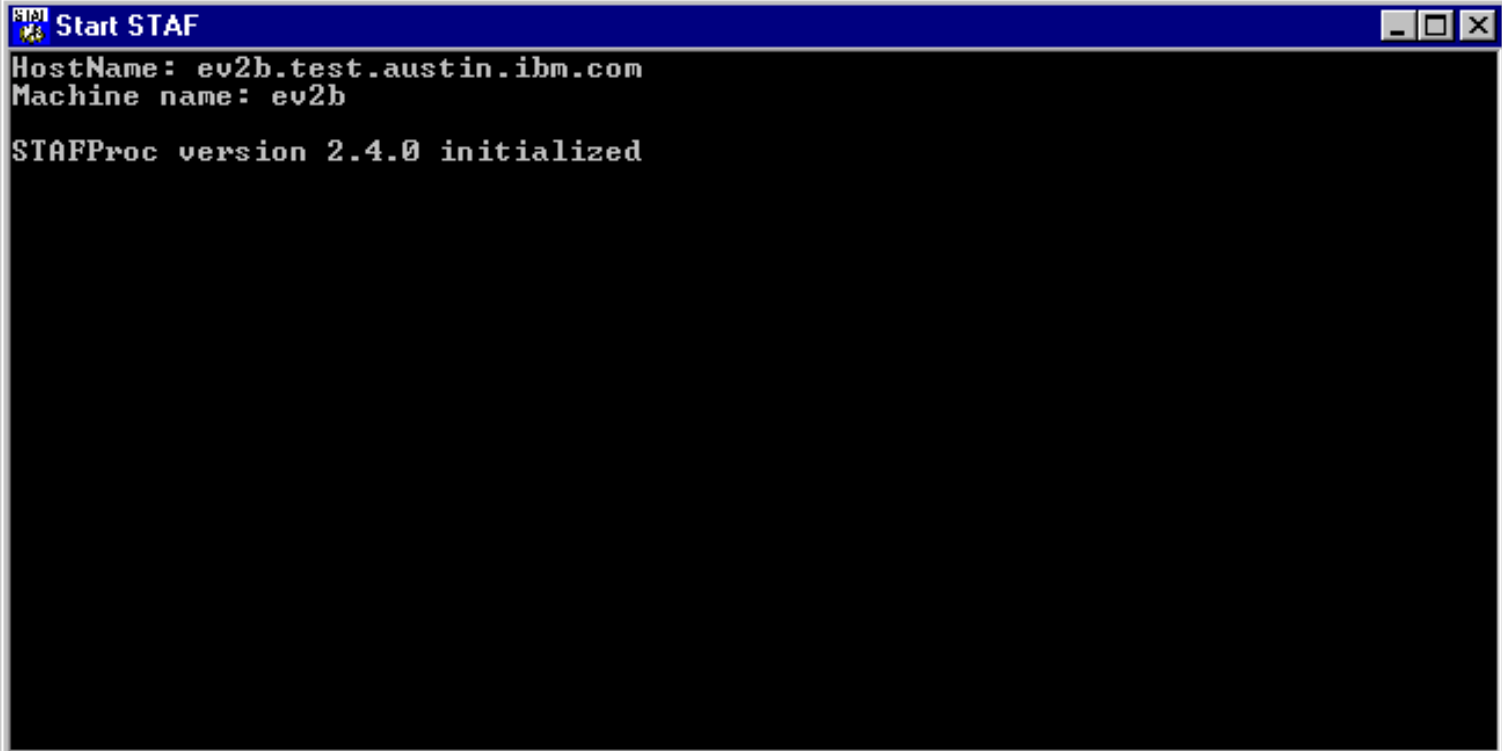
STAF COMMANDS

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 "ev2b" (our local machine) and "ev2c" (our remote machine). You will need to substitute your machines names.

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 - Software Testing Automation Framework", 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:



```
Start STAF
HostName: ev2b.test.austin.ibm.com
Machine name: ev2b

STAFProc version 2.4.0 initialized
```

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

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

Note: 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.

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.

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 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 (see "Submitting STAF Requests" in the [BASIC STAF CONCEPTS](#) Topic for more information).

The syntax of this command is:

```
STAF <Where> <Service> <Request>
```

<Where> is the machine name to where the request will be submitted

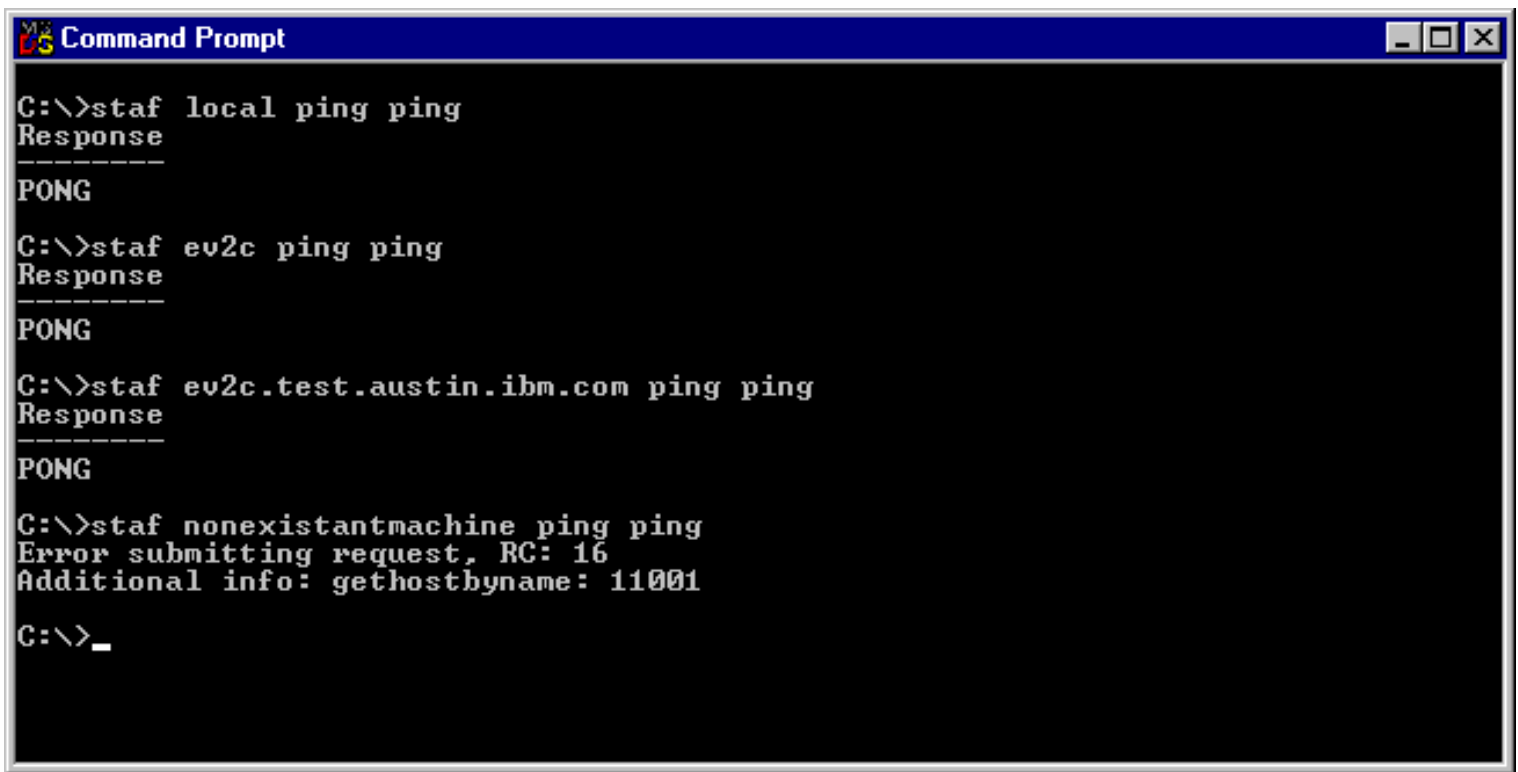
<Service> is the name of the service that will receive and process the request

<Request> is the service request

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.

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:

A screenshot of a Windows Command Prompt window titled "Command Prompt". The window has a blue title bar with standard Windows window controls (minimize, maximize, close). The background is black with white text. The text shows four STAF commands being executed in a C:\ directory. Each command is followed by a "Response" line and a "PONG" message, except for the last one which shows an error. The commands are: 1. "staf local ping ping" resulting in "PONG". 2. "staf ev2c ping ping" resulting in "PONG". 3. "staf ev2c.test.austin.ibm.com ping ping" resulting in "PONG". 4. "staf nonexistentmachine ping ping" resulting in an error: "Error submitting request, RC: 16" and "Additional info: gethostbyname: 11001". The prompt ends with "C:\>_".

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

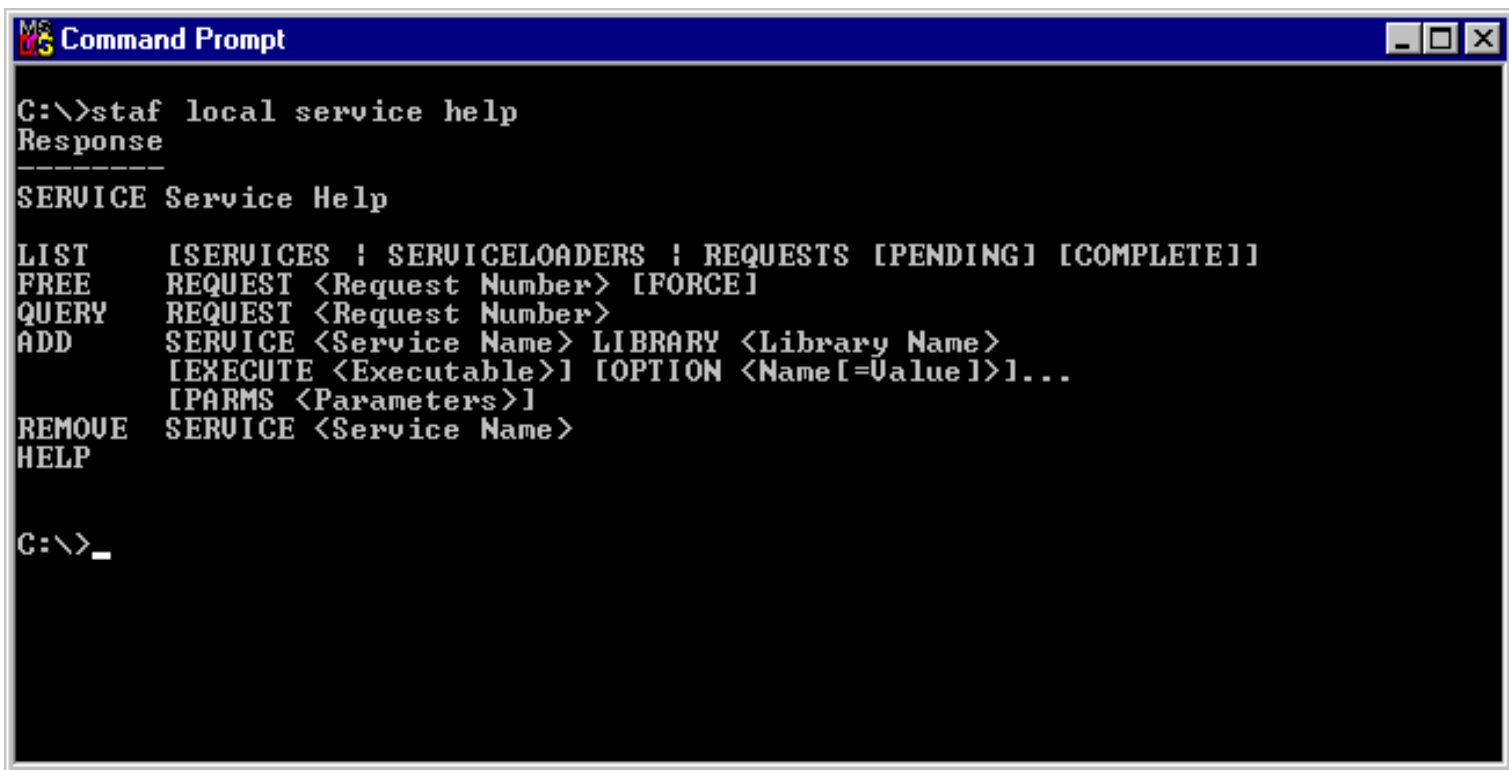
The second command, **STAF ev2c ping ping**, demonstrates that you can also use the short hostname (ev2c) for the remote machine.

The third command, **STAF ev2c.test.austin.ibm.com ping ping**, pings a remote machine using its full hostname (ev2c.test.austin.ibm.com).

The fourth command, **STAF nonexistentmachine ping ping**, fails because remote machine nonexistentmachine cannot be found.

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:



```

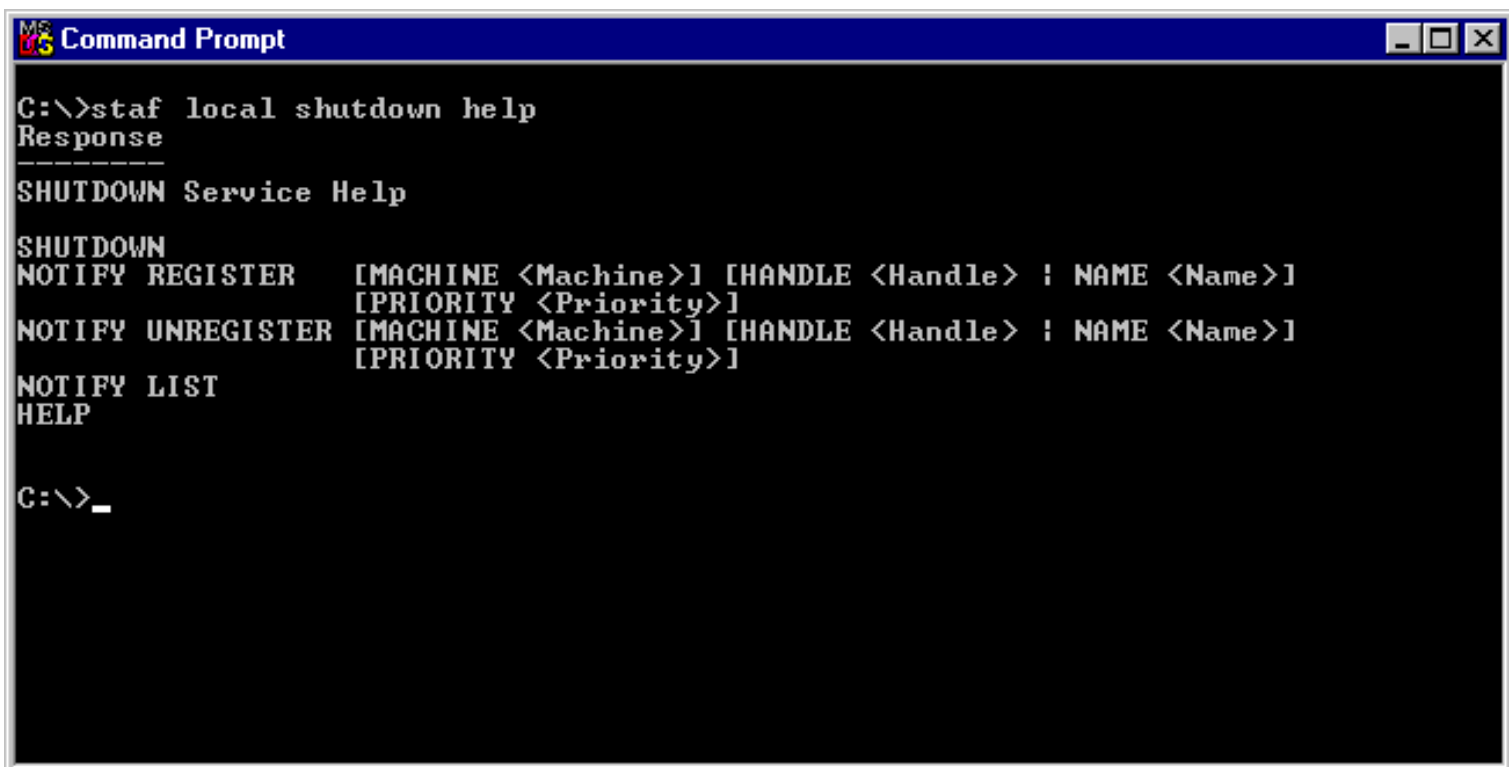
MS-DOS Command Prompt
C:\>staf local service help
Response
-----
SERVICE Service Help

LIST      [SERVICES ; SERVICELOADERS ; REQUESTS [PENDING] [COMPLETE]]
FREE     REQUEST <Request Number> [FORCE]
QUERY   REQUEST <Request Number>
ADD      SERVICE <Service Name> LIBRARY <Library Name>
        [EXECUTE <Executable>] [OPTION <Name [=Value]>]...
        [PARMS <Parameters>]
REMOVE  SERVICE <Service Name>
HELP

C:\>_

```

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:



```

MS-DOS Command Prompt
C:\>staf local shutdown help
Response
-----
SHUTDOWN Service Help

SHUTDOWN
NOTIFY REGISTER  [MACHINE <Machine>] [HANDLE <Handle> ; NAME <Name>]
                [PRIORITY <Priority>]
NOTIFY UNREGISTER [MACHINE <Machine>] [HANDLE <Handle> ; NAME <Name>]
                [PRIORITY <Priority>]
NOTIFY LIST
HELP

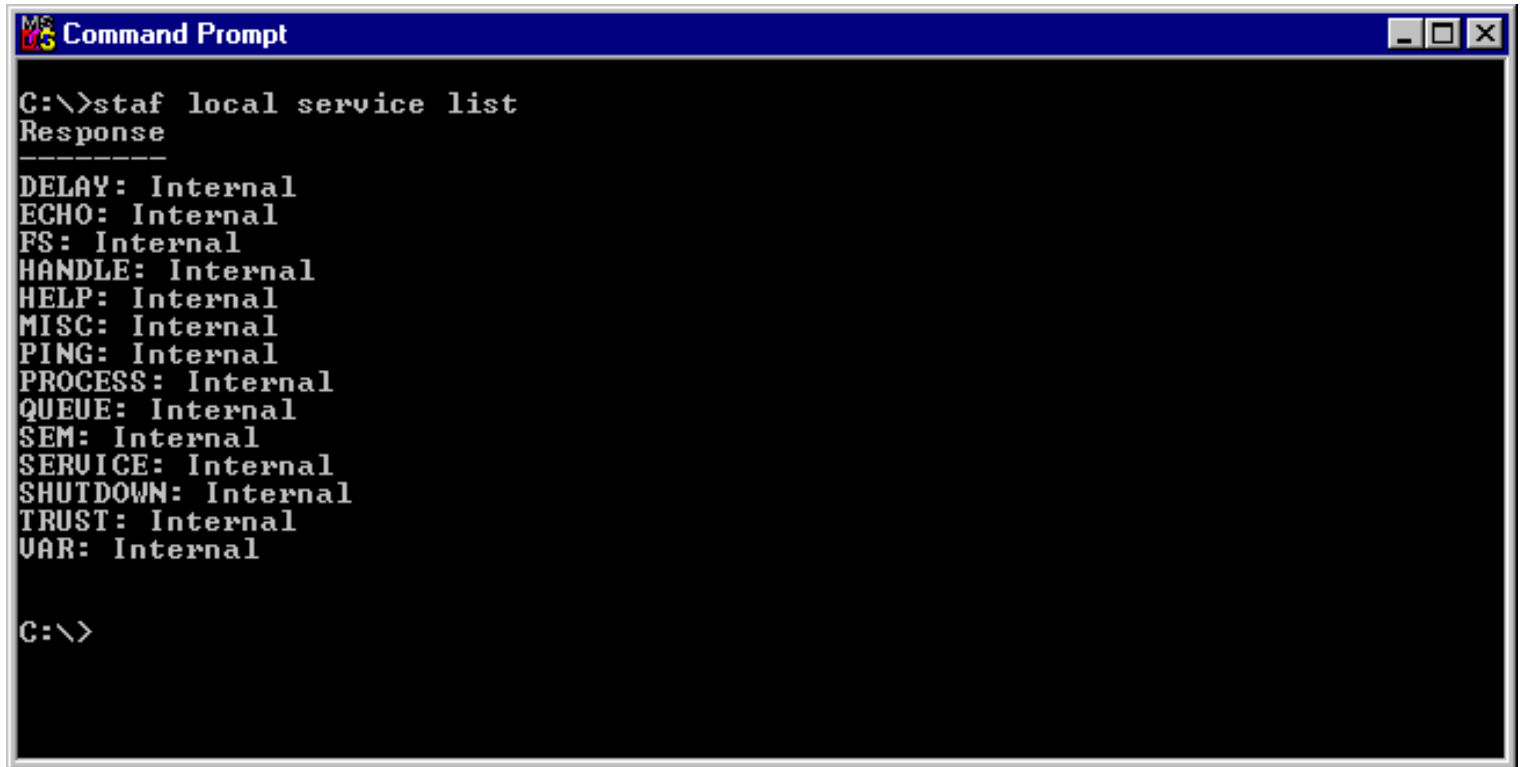
C:\>_

```

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.

Listing available Services

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

A screenshot of a Windows Command Prompt window. The title bar reads "MS-DOS Command Prompt". The command prompt shows the command "C:\>staf local service list" and the output "Response" followed by a list of services: DELAY: Internal, ECHO: Internal, FS: Internal, HANDLE: Internal, HELP: Internal, MISC: Internal, PING: Internal, PROCESS: Internal, QUEUE: Internal, SEM: Internal, SERVICE: Internal, SHUTDOWN: Internal, TRUST: Internal, and VAR: Internal. The prompt "C:\>" is visible at the bottom.

```
MS-DOS Command Prompt
C:\>staf local service list
Response
-----
DELAY: Internal
ECHO: Internal
FS: Internal
HANDLE: Internal
HELP: Internal
MISC: Internal
PING: Internal
PROCESS: Internal
QUEUE: Internal
SEM: Internal
SERVICE: Internal
SHUTDOWN: Internal
TRUST: Internal
VAR: Internal

C:\>
```

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.

Listing Variables

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

```

MS-DOS Command Prompt
Response
-----
STAF/Config/BootDrive=C:
STAF/Config/ConfigFile=C:\STAF\bin\STAF.cfg
STAF/Config/EffectiveMachine=ev2b
STAF/Config/Machine=ev2b.test.austin.ibm.com
STAF/Config/Mem/Physical/Bytes=267710464
STAF/Config/Mem/Physical/KB=261436
STAF/Config/Mem/Physical/MB=255
STAF/Config/OS/MajorVersion=4
STAF/Config/OS/MinorVersion=0
STAF/Config/OS/Name=WinNT
STAF/Config/OS/Revision=1381
STAF/Config/Sep/File=\
STAF/Config/Sep/Line=

STAF/Config/Sep/Path=;
STAF/Config/STAFRoot=C:\STAF
STAF/Env/ClassPath=C:\STAF\bin;C:\STAF\bin\JSTAF.zip;C:\STAF\samples\demo\STAFDemo.jar;.
STAF/Env/COMPUTERNAME=EU2B
STAF/Env/ComSpec=C:\WINNT\system32\cmd.exe
STAF/Env/HOMEDRIVE=C:
-- More --

```

```

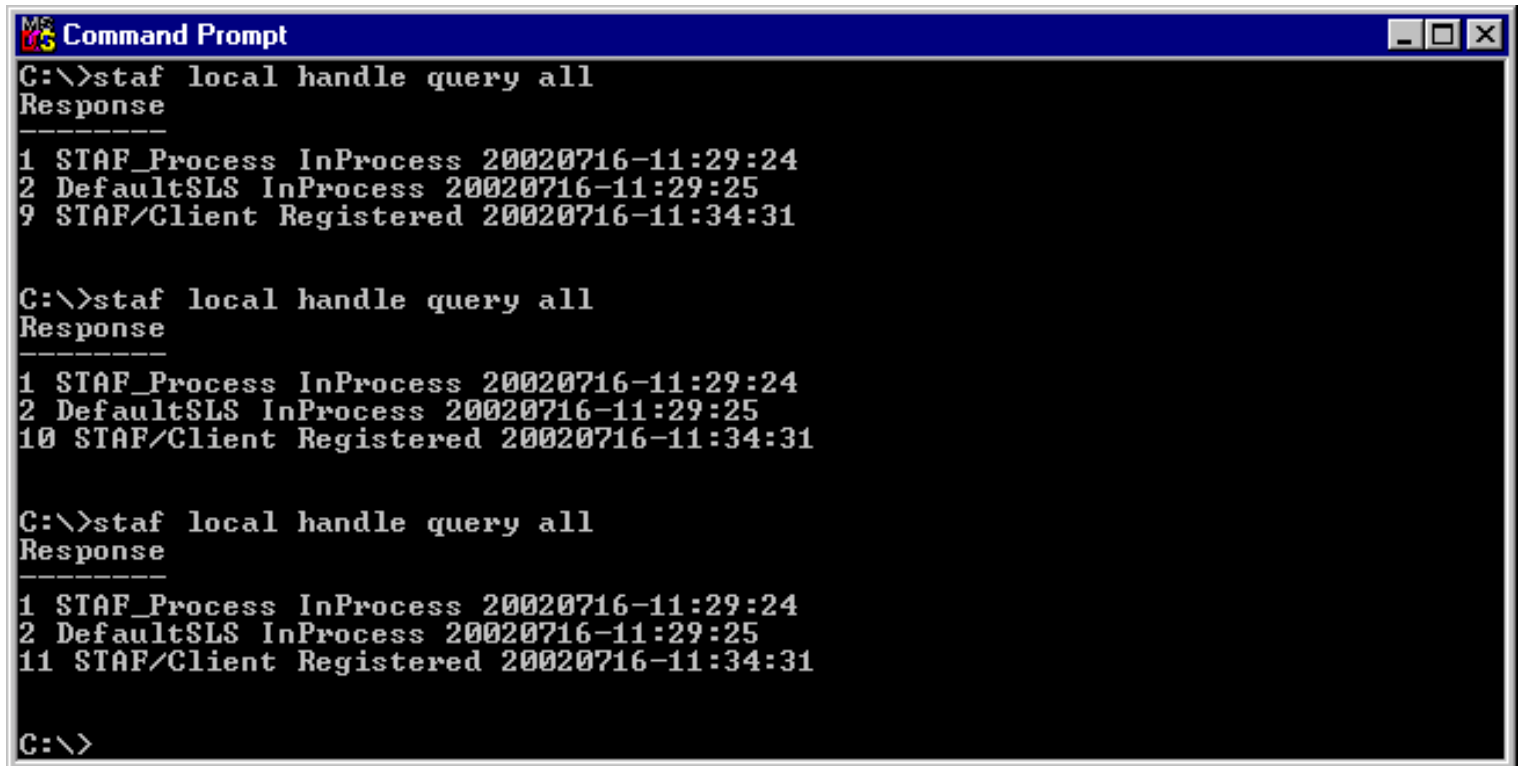
MS-DOS Command Prompt
STAF/Env/HOMEPATH=\
STAF/Env/INCLUDE=C:\Program Files\ObjREXX\API
STAF/Env/LIB=C:\Program Files\ObjREXX\API
STAF/Env/LOGONSERUER=\\EU2B
STAF/Env/NUMBER_OF_PROCESSORS=1
STAF/Env/OS=Windows_NT
STAF/Env/Os2LibPath=C:\WINNT\system32\os2\dll;
STAF/Env/Path=C:\STAF\bin;C:\STAF\bin\java12;c:\jdk1.3.1_01\jre\bin;c:\jdk1.3.1_01\bin;;C:\WINNT\system32;C:\WINNT;C:\Program Files\ObjREXX;C:\Program Files\ObjREXX\OODIALOG;
STAF/Env/PROCESSOR_ARCHITECTURE=x86
STAF/Env/PROCESSOR_IDENTIFIER=x86 Family 6 Model 8 Stepping 3, GenuineIntel
STAF/Env/PROCESSOR_LEVEL=6
STAF/Env/PROCESSOR_REVISION=0803
STAF/Env/SOUNDPATH=C:\WINNT
STAF/Env/STAFCONUDIR=C:\STAF\codepage
STAF/Env/SystemDrive=C:
STAF/Env/SystemRoot=C:\WINNT
STAF/Env/TEMP=C:\TEMP
STAF/Env/TMP=C:\TEMP
STAF/Env/USERDOMAIN=EU2B
STAF/Env/USERNAME=Administrator
STAF/Env/USERPROFILE=C:\WINNT\Profiles\Administrator
-- More --

```


Notice that the second command (**staf ev4a var global get STAF/Config/Sep/File**) was sent to a machine running a Unix operating system, so the output reflects the Unix File Separator.

Querying Handles

To query the current STAF handles, issue the following command: **STAF local handle query all**



```

C:\>staf local handle query all
Response
-----
1 STAF_Process InProcess 20020716-11:29:24
2 DefaultSLS InProcess 20020716-11:29:25
9 STAF/Client Registered 20020716-11:34:31

C:\>staf local handle query all
Response
-----
1 STAF_Process InProcess 20020716-11:29:24
2 DefaultSLS InProcess 20020716-11:29:25
10 STAF/Client Registered 20020716-11:34:31

C:\>staf local handle query all
Response
-----
1 STAF_Process InProcess 20020716-11:29:24
2 DefaultSLS InProcess 20020716-11:29:25
11 STAF/Client Registered 20020716-11:34:31

C:\>

```

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 query all" 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 9 and 10).

CONFIGURING STAF

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.

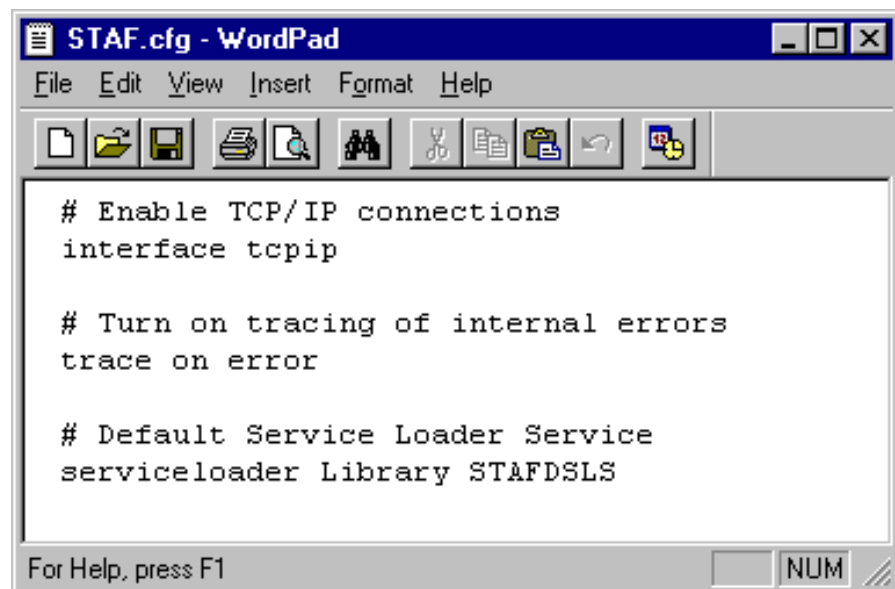
When STAFProc is started on a machine, that machine's STAF.cfg file will be read to determine how STAF should be configured on the machine.

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.

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:



Notice that comments start with **#**.

The first configuration statement is **interface tcpip**. 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. Note that all machines in the same STAF environment must use the same port. 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 7500, the statement would be: **interface tcpip 7500**.

To work in a disconnected mode, instead of the Interface statement you would use the Machine statement followed by the name by which you want your machine to be known. For example, to work in a disconnected mode with the name "TestMachine" for your machine, the statement would be: **machine TestMachine**.

Note that the Interface and Machine configuration statements are mutually exclusive.

The **trace on error** statement causes a trace message to be generated for error conditions that STAF detects, such as broken communication connections and fatal service errors.

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

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

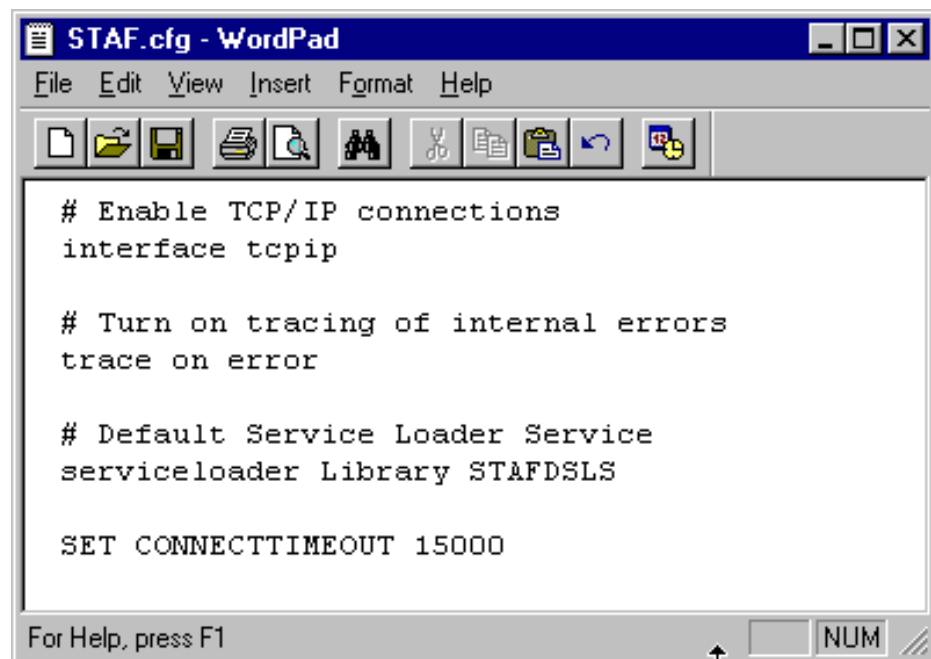
Operational Parameters

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 `CONNECTTIMEOUT`. It specifies the maximum time in milliseconds to wait for a connection to a remote system to succeed. The default is 5000 (i.e., 5 seconds). If you wanted to increase the time to 15 seconds, the statement would be:

SET CONNECTTIMEOUT 15000 Add this statement to your STAF.cfg file and save it. Remember to shutdown and restart STAFProc to pick up the STAF.cfg updates.

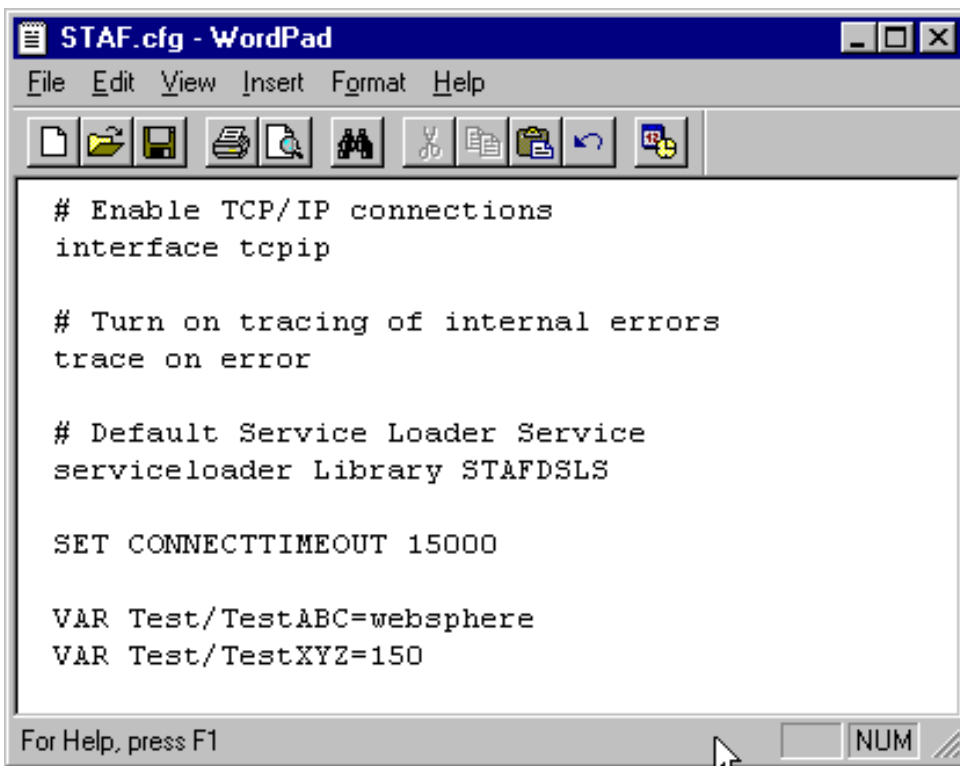


Make sure that your remote STAF machine is up and running but does NOT have STAFProc currently running. Now try the **STAF ev2c ping ping** command again from your machine which is running STAFProc, except substitute your remote machine's hostname for "ev2c". Notice that it takes 15 seconds for the timeout to occur.

Setting Variables

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

VAR Test/TestABC=websphere
VAR Test/TestXYZ=150



```

# Enable TCP/IP connections
interface tcpip

# Turn on tracing of internal errors
trace on error

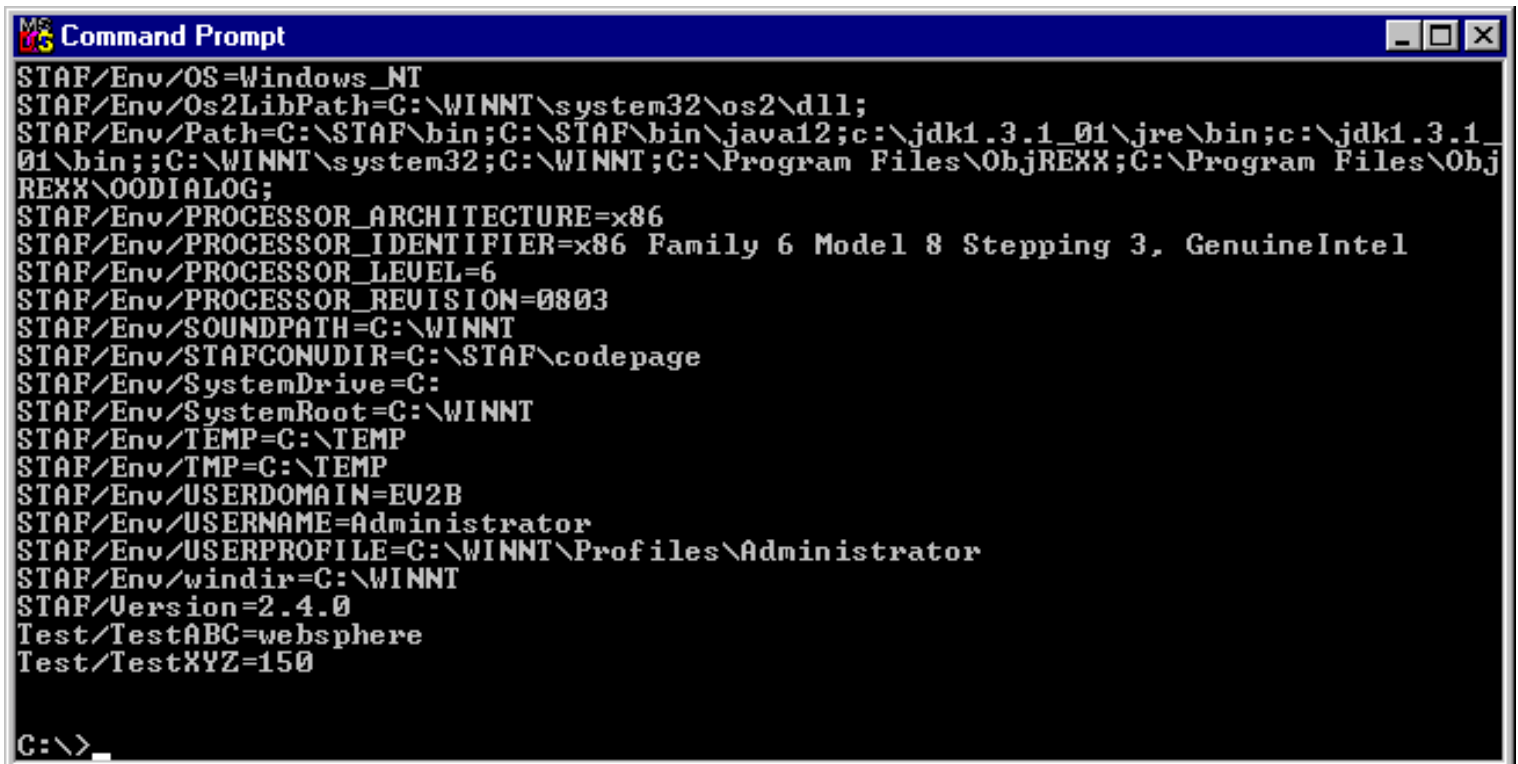
# Default Service Loader Service
serviceloader Library STAFDSLS

SET CONNECTTIMEOUT 15000

VAR Test/TestABC=websphere
VAR Test/TestXYZ=150

```

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



```

STAF/Env/OS=Windows_NT
STAF/Env/Os2LibPath=C:\WINNT\system32\os2\dll;
STAF/Env/Path=C:\STAF\bin;C:\STAF\bin\java12;c:\jdk1.3.1_01\jre\bin;c:\jdk1.3.1_01\bin;;C:\WINNT\system32;C:\WINNT;C:\Program Files\ObjREXX;C:\Program Files\ObjREXX\OODIALOG;
STAF/Env/PROCESSOR_ARCHITECTURE=x86
STAF/Env/PROCESSOR_IDENTIFIER=x86 Family 6 Model 8 Stepping 3, GenuineIntel
STAF/Env/PROCESSOR_LEVEL=6
STAF/Env/PROCESSOR_REVISION=0803
STAF/Env/SOUNDPATH=C:\WINNT
STAF/Env/STAFCONUDIR=C:\STAF\codepage
STAF/Env/SystemDrive=C:
STAF/Env/SystemRoot=C:\WINNT
STAF/Env/TEMP=C:\TEMP
STAF/Env/TMP=C:\TEMP
STAF/Env/USERDOMAIN=EU2B
STAF/Env/USERNAME=Administrator
STAF/Env/USERPROFILE=C:\WINNT\Profiles\Administrator
STAF/Env/windir=C:\WINNT
STAF/Version=2.4.0
Test/TestABC=websphere
Test/TestXYZ=150

C:\>_

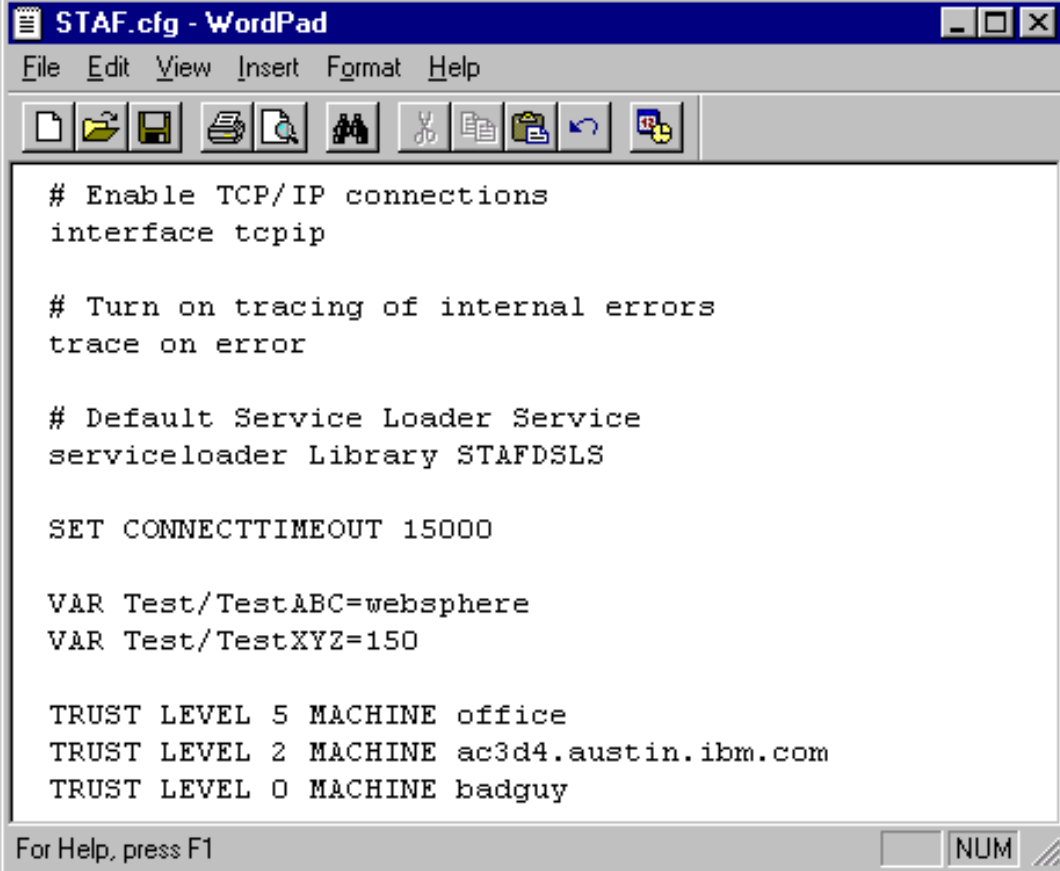
```

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

Trust Levels

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

TRUST LEVEL 5 MACHINE office
TRUST LEVEL 2 MACHINE ac3d4.austin.ibm.com
TRUST LEVEL 0 MACHINE badguy



```

# Enable TCP/IP connections
interface tcpip

# Turn on tracing of internal errors
trace on error

# Default Service Loader Service
serviceloader Library STAFDSLS

SET CONNECTTIMEOUT 15000

VAR Test/TestABC=websphere
VAR Test/TestXYZ=150

TRUST LEVEL 5 MACHINE office
TRUST LEVEL 2 MACHINE ac3d4.austin.ibm.com
TRUST LEVEL 0 MACHINE badguy

```

Now restart STAFProc on your machine.

The numeric trust levels are defined in the STAF User's Guide; higher numbers indicate greater access. A trust level of 5 gives machine "office" the highest level of access to your machine. A trust level of 2 gives machine "ac3d4.austin.ibm.com" a limited level of access to your machine. A trust level of 0 gives machine "badguy" no access to your machine.

You can set the default trust level by adding this line to your STAF.cfg file:

TRUST DEFAULT LEVEL 4

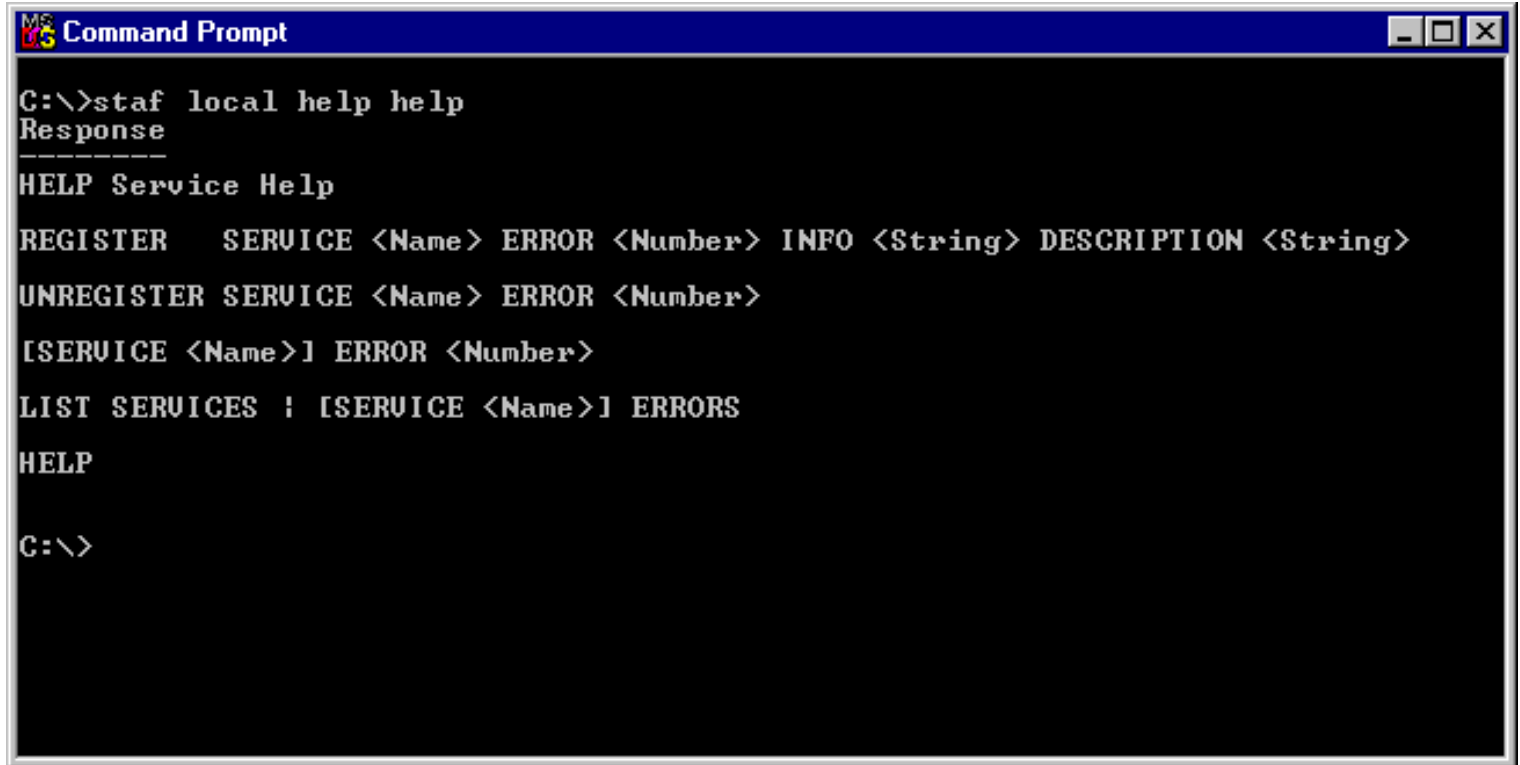
Setting the default trust level to 4 indicates that all machines in the STAF Environment that are not specified in other TRUST configuration statements will have a trust level of 4. If you do not specify a default trust level in your STAF.cfg file, the default is set to 3.

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 the 3 TRUST entries from your STAF.cfg file and restart STAFProc.

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**

A screenshot of a Windows Command Prompt window. The title bar reads "Command Prompt" and includes standard window controls (minimize, maximize, close). The command prompt shows the command "C:\>staf local help help" and the following output:

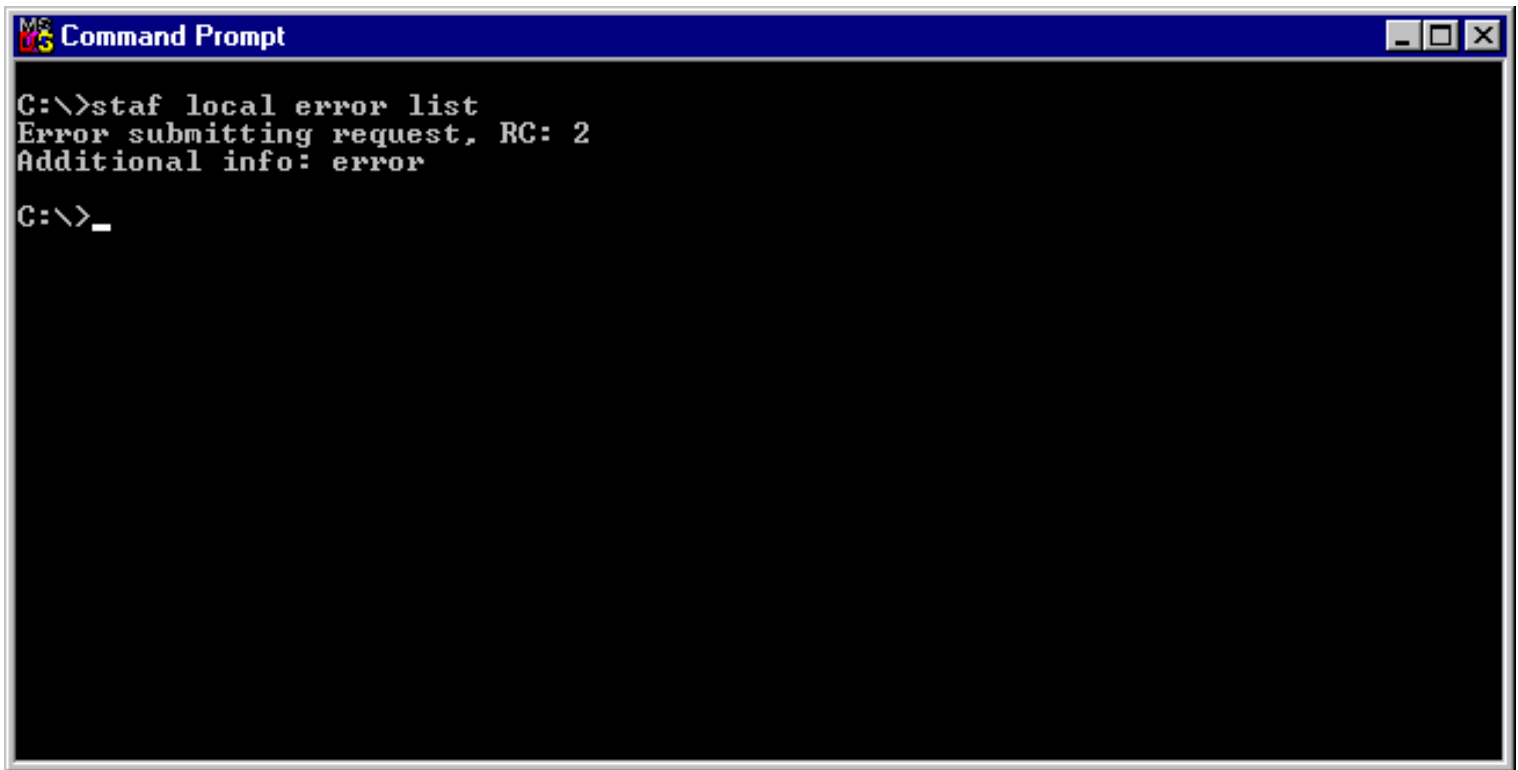
```
Response
-----
HELP Service Help

REGISTER    SERVICE <Name> ERROR <Number> INFO <String> DESCRIPTION <String>
UNREGISTER SERVICE <Name> ERROR <Number>
[SERVICE <Name>] ERROR <Number>
LIST SERVICES : [SERVICE <Name>] ERRORS
HELP

C:\>
```

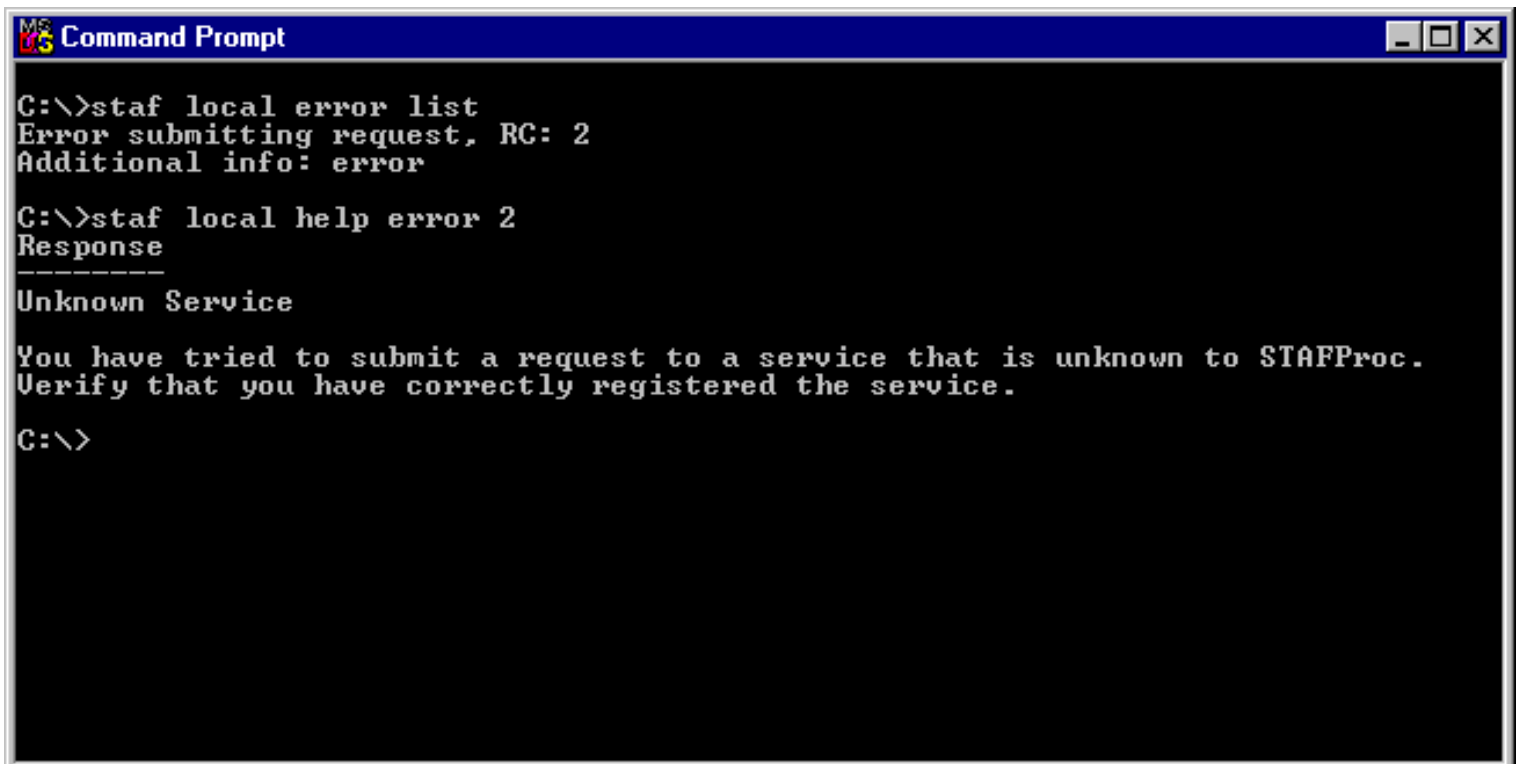
To demonstrate the STAF Help Service, type the following command (note that we are intentionally specifying an invalid service name "error"):

STAF local error list



```
MS-DOS Command Prompt
C:\>staf local error list
Error submitting request, RC: 2
Additional info: error
C:\>_
```

To find out what a RC 2 means, issue the following command: [STAF local help error 2](#)



```
MS-DOS Command Prompt
C:\>staf local error list
Error submitting request, RC: 2
Additional info: error
C:\>staf local help error 2
Response
-----
Unknown Service

You have tried to submit a request to a service that is unknown to STAFProc.
Verify that you have correctly registered the service.
C:\>
```

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

REGISTERING 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.

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) or Java Runtime Environment (JRE) installed on your test machines.

While most of the existing Java STAF Services require Java 1.1 or higher, the STAF Demo requires Java 1.2 or higher, so it is recommended that you install Java 1.2 or higher on both of your STAF machines.

You can obtain the Sun Java Standard Edition versions from its web site: <http://java.sun.com/j2se/>

You can download the IBM Java versions from: <http://www-106.ibm.com/developerworks/java/jdk/index.html>

IBM Employees can obtain the IBM Java versions from the IBM Intranet at:
<http://w3.hursley.ibm.com/java/JIM.html>

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:\jdk1.3.1\jre\bin (assuming that you installed Java 1.3.1 to directory c:\jdk1.3.1) must be in your System PATH.

Using REXX STAF Services

In order to run a Rexx STAF Service, you will need to have Object Rexx installed on your system. You can find information about REXX at the following web site: http://www.mindspring.com/~dave_martin/RexxFAQ.html

IBM Employees can download Object Rexx from the IBM Intranet at:
<ftp://sdfrxs02.boeblingen.de.ibm.com/rexx>.

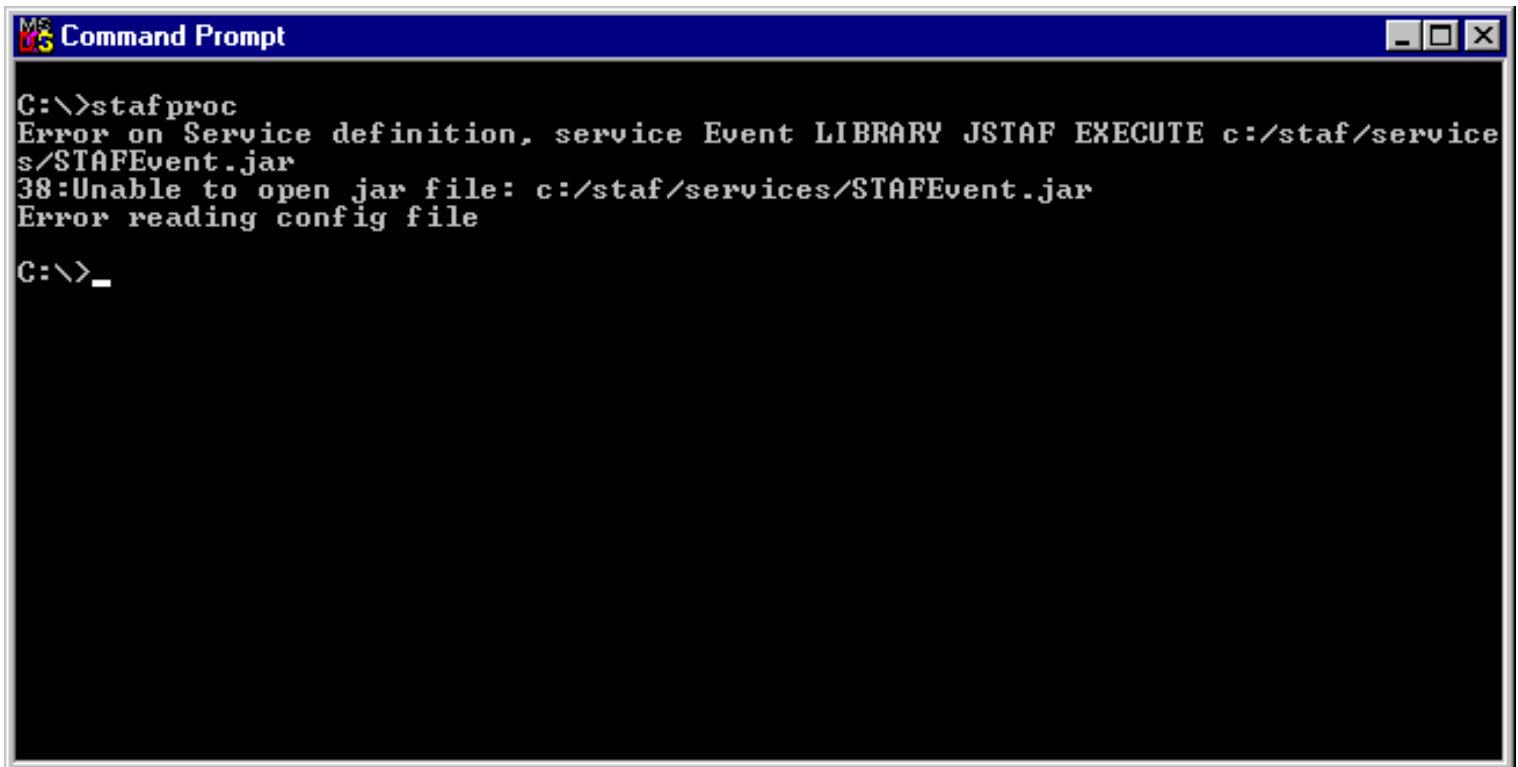
Registering Services

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 fully-qualified path to 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:

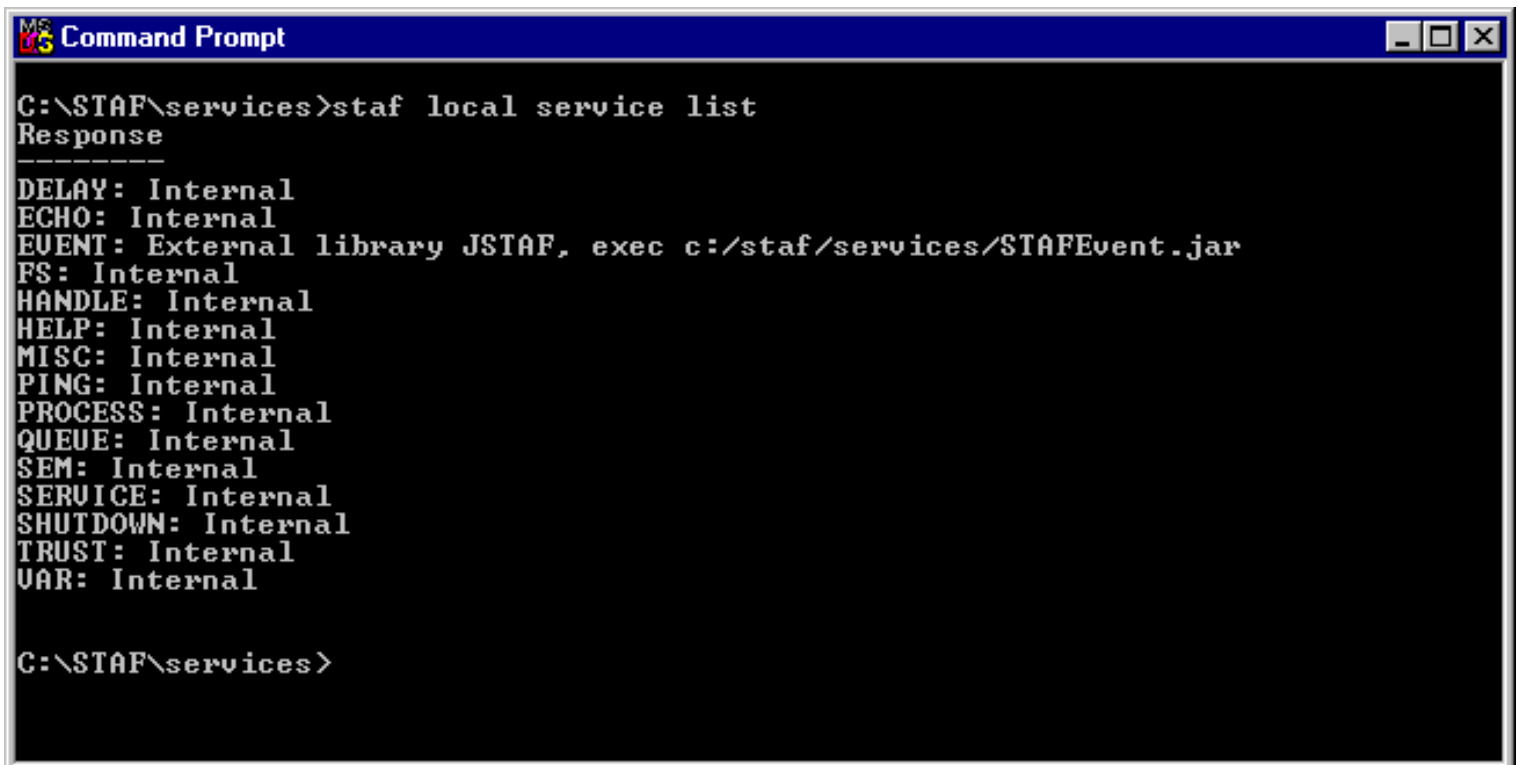


```

MS-DOS Command Prompt
C:\>stafproc
Error on Service definition, service Event LIBRARY JSTAF EXECUTE c:/staf/service
s/STAFEvent.jar
38:Unable to open jar file: c:/staf/services/STAFEvent.jar
Error reading config file
C:\>_

```

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:



```

MS-DOS Command Prompt
C:\STAF\services>staf local service list
Response
-----
DELAY: Internal
ECHO: Internal
EUEENT: External library JSTAF, exec c:/staf/services/STAFEvent.jar
FS: Internal
HANDLE: Internal
HELP: Internal
MISC: Internal
PING: Internal
PROCESS: Internal
QUEUE: Internal
SEM: Internal
SERVICE: Internal
SHUTDOWN: Internal
TRUST: Internal
UAR: Internal

C:\STAF\services>

```

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.

STAF DEMO - Running the 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 machine (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:


```

# Enable TCP/IP connections
interface tcpip

# Turn on tracing of internal errors
trace on error

# Default Service Loader Service
serviceloader Library STAFDSLS

SET CONNECTTIMEOUT 15000

VAR Test/TestABC=websphere
VAR Test/TestXYZ=150

service Event LIBRARY JSTAF EXECUTE c:/staf/services/STAFEvent.jar

TRUST LEVEL 5 MACHINE ev2c

```

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

```

# Enable TCP/IP connections
interface tcpip

# Turn on tracing of internal errors
trace on error

# Default Service Loader Service
serviceloader Library STAFDefaultSLS

trust level 5 machine ev2b

```

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

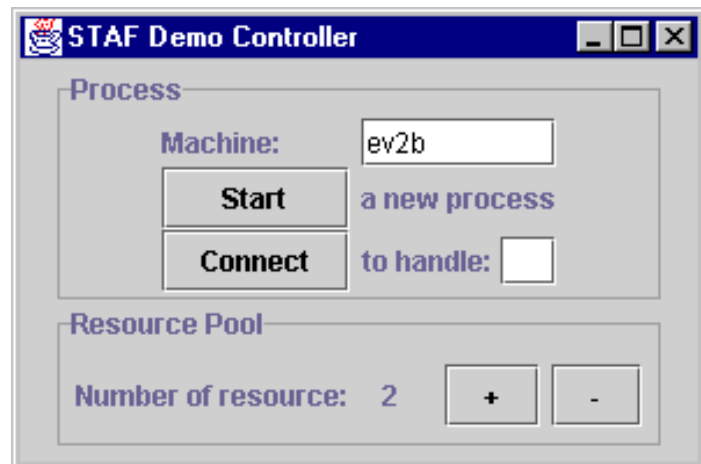
- STAFDemoController.java
- STAFProcess.java

Starting the STAF Demo

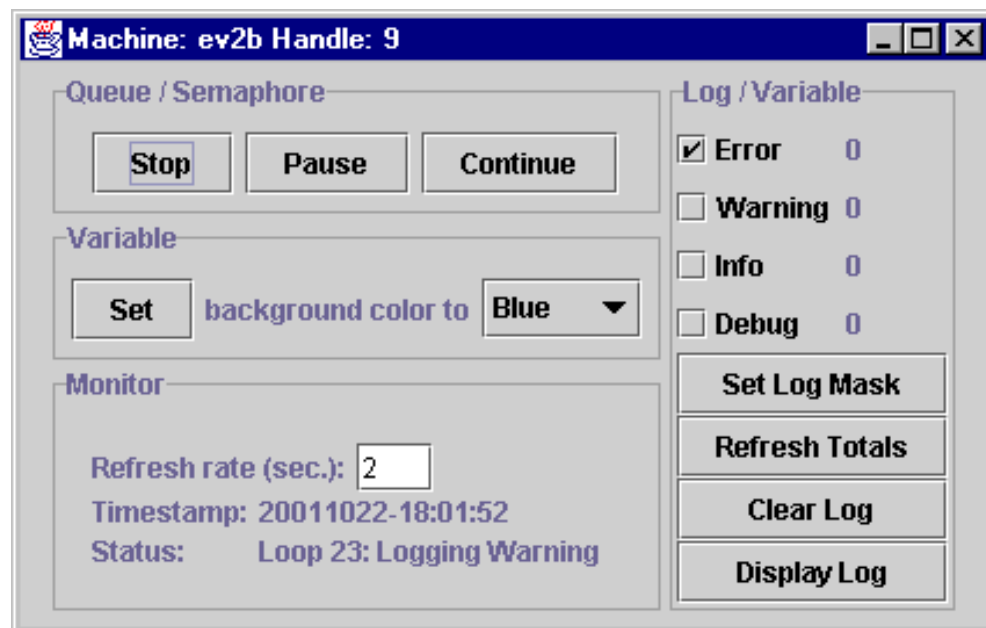
At this point you should have the STAF Demo set up on both of your machines, so let's start the demo. First, let's run it locally on one machine. Make sure STAFProc is up and running with your latest STAF.cfg updates. From a command prompt, enter:

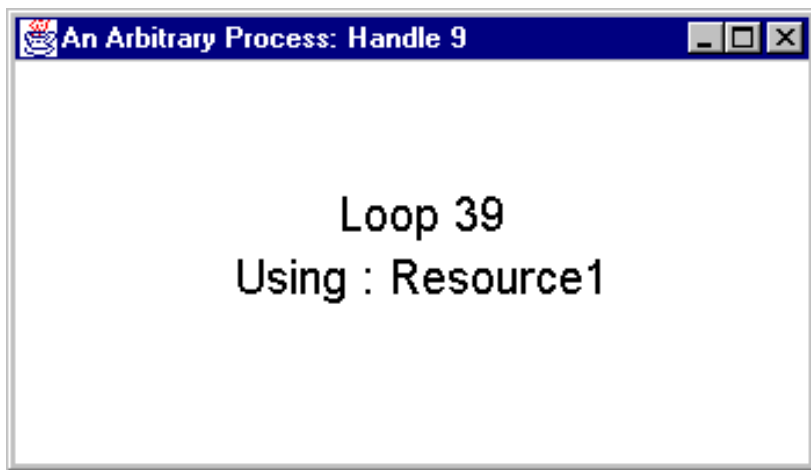
java STAFDemoController

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



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.



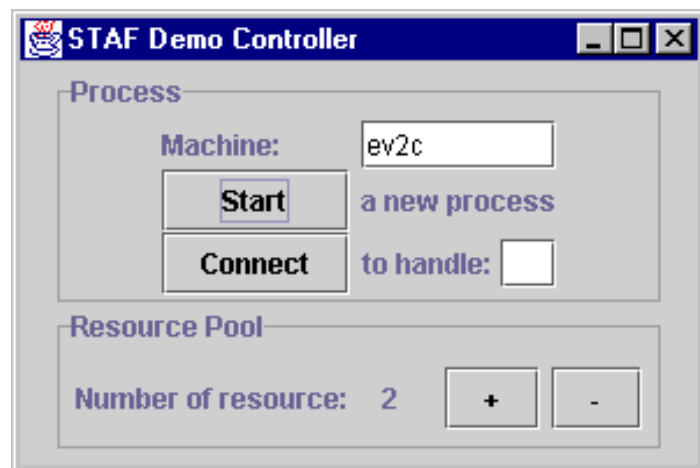


The first dialog ("Machine:....") is the Control window, and the second dialog ("An Arbitrary Process...") is the sample application (STAFProcess). The sample application will loop indefinitely. Note that the titles of the sections in the Control window ("Queue/Semaphore", "Variable", "Monitor", "Log/Variable")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

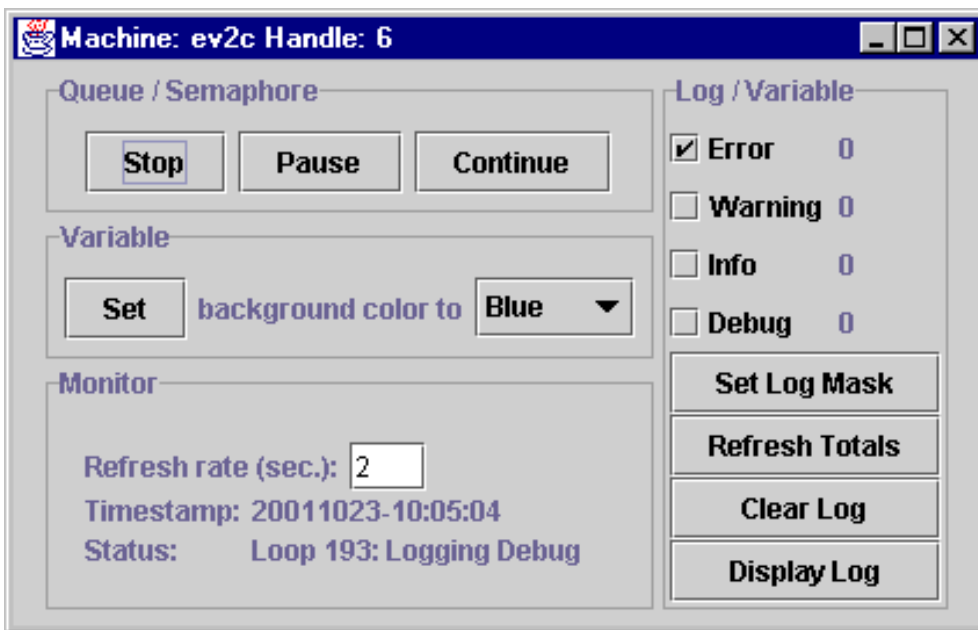
Click on the Stop button in the Control window (this will cause the "An Arbitrary Process" window to close), and then close the Control window.

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.

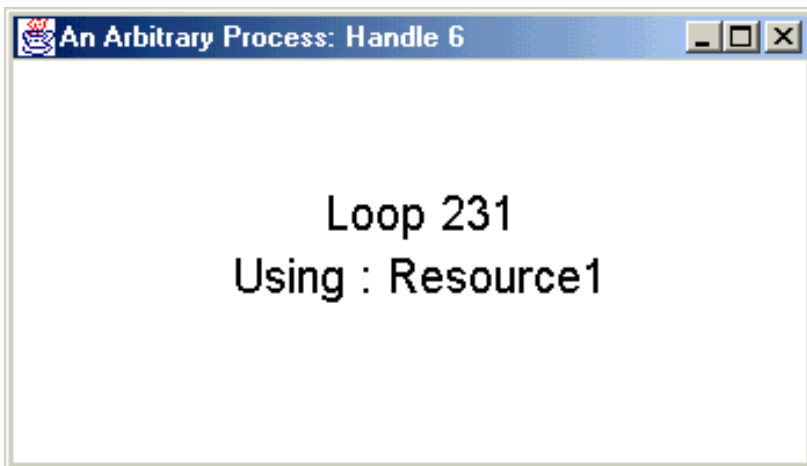
In the STAF Demo Controller window, change "ev2b" to your remote machine (for the tutorial this will be "ev2c"):



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

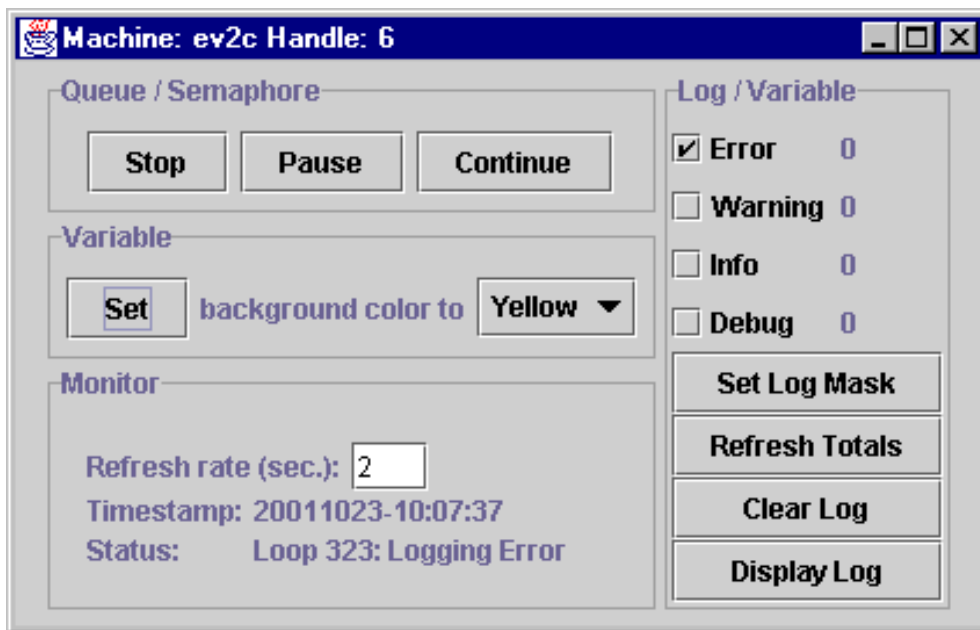


However, the STAFProcess window should be displayed on your remote machine:

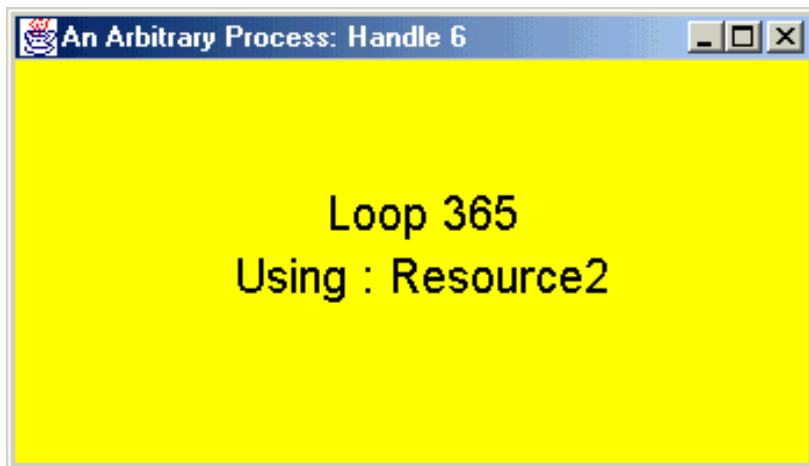


Using the Variable service to change the application's background color

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:



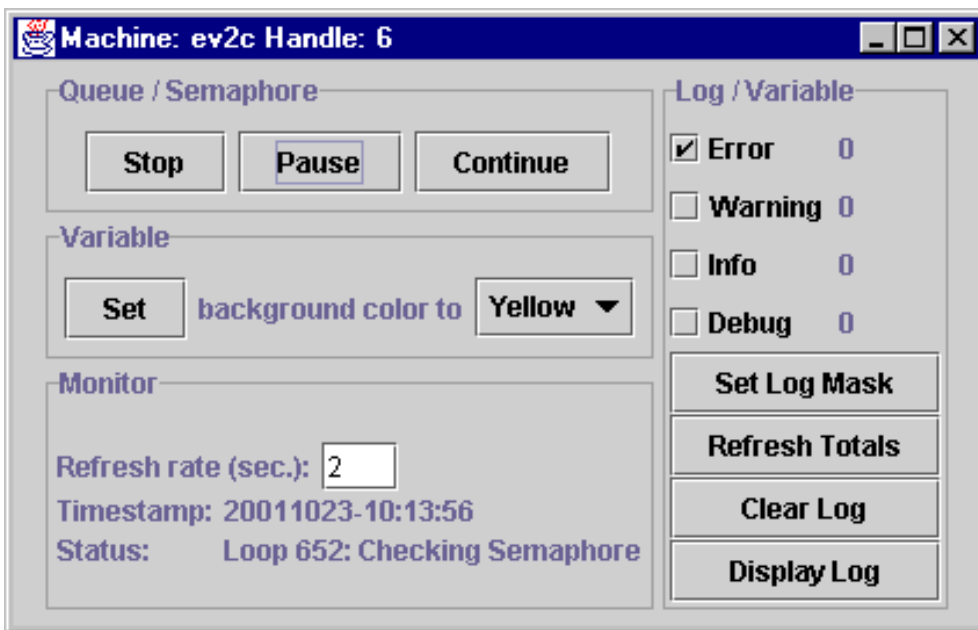
You should see the update to the sample application on your remote system:



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.

Using the Semaphore Service to Control the Application's Execution

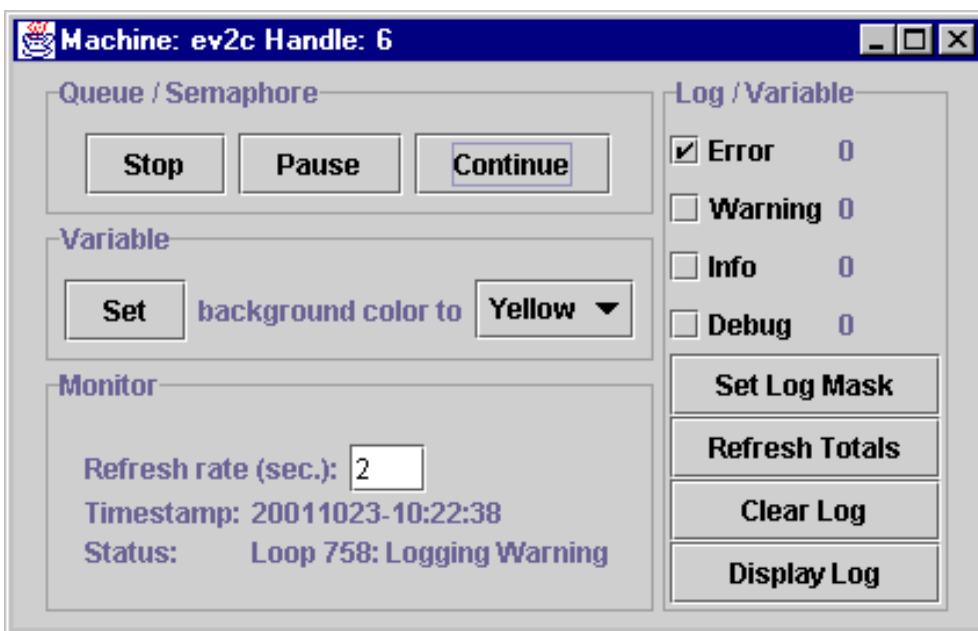
Click the "Pause" button in the Control window:



You should see that the application on the remote system is no longer be 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.

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:

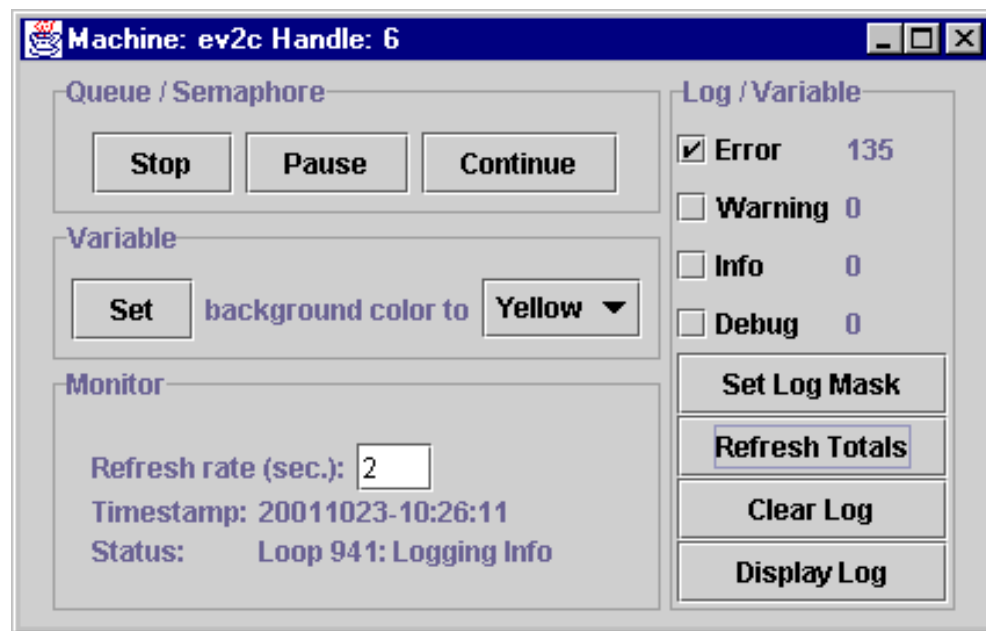


You should see the information updated every few seconds. Note, you can determine where, in execution, the application is, without needing to actually look at the application display. This is particularly useful when the systems running the applications are not "conveniently" located. In order to do this, the application simply logs

occasional messages via the Monitor service.

Using the Log service

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



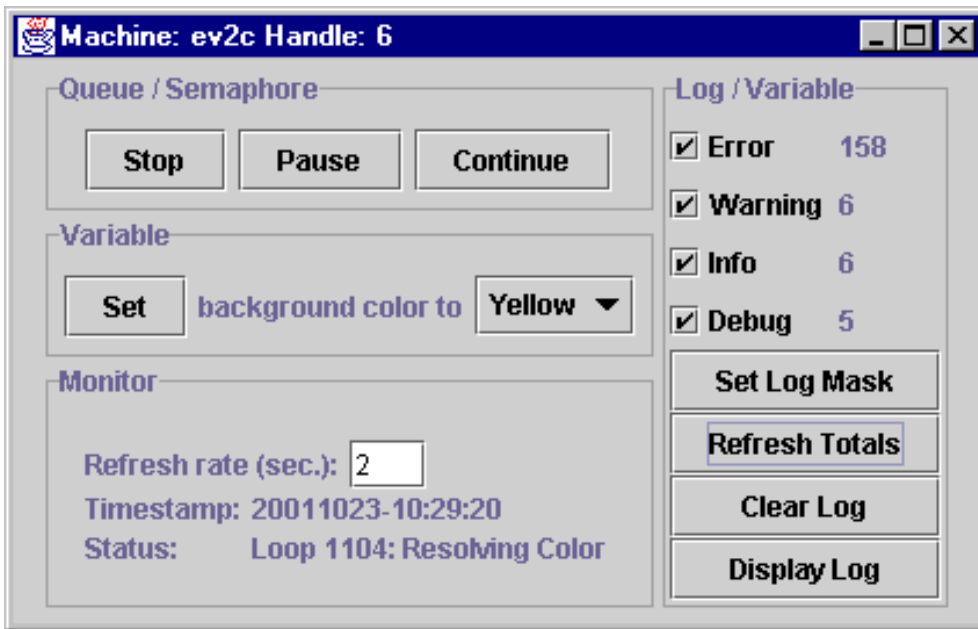
Click "Display Log" and you should see a new window which contains all the logged error messages.

Timestamp	Level	Message
20011023-10:01:24	Error	Loop 1: Error Message
20011023-10:01:32	Error	Loop 8: Error Message
20011023-10:01:40	Error	Loop 15: Error Message
20011023-10:01:48	Error	Loop 22: Error Message
20011023-10:01:56	Error	Loop 29: Error Message
20011023-10:02:04	Error	Loop 36: Error Message
20011023-10:02:12	Error	Loop 43: Error Message
20011023-10:02:21	Error	Loop 50: Error Message
20011023-10:02:29	Error	Loop 57: Error Message
20011023-10:02:37	Error	Loop 64: Error Message
20011023-10:02:45	Error	Loop 71: Error Message
20011023-10:02:53	Error	Loop 78: Error Message
20011023-10:03:01	Error	Loop 85: Error Message
20011023-10:03:09	Error	Loop 92: Error Message
20011023-10:03:17	Error	Loop 99: Error Message
20011023-10:03:25	Error	Loop 106: Error Message

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



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.



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

Timestamp	Level	Message
20011023-10:30:13	Error	Loop 1149: Error Message
20011023-10:30:14	Warning	Loop 1150: Warning Message
20011023-10:30:15	Info	Loop 1151: Info Message
20011023-10:30:16	Debug	Loop 1152: Debug message
20011023-10:30:21	Error	Loop 1156: Error Message
20011023-10:30:22	Warning	Loop 1157: Warning Message
20011023-10:30:23	Info	Loop 1158: Info Message
20011023-10:30:24	Debug	Loop 1159: Debug message
20011023-10:30:29	Error	Loop 1163: Error Message
20011023-10:30:30	Warning	Loop 1164: Warning Message
20011023-10:30:31	Info	Loop 1165: Info Message
20011023-10:30:32	Debug	Loop 1166: Debug message
20011023-10:30:37	Error	Loop 1170: Error Message
20011023-10:30:38	Warning	Loop 1171: Warning Message
20011023-10:30:39	Info	Loop 1172: Info Message
20011023-10:30:40	Debug	Loop 1173: Debug message

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.

Machine: ev2c Handle: 6

Queue / Semaphore

Stop Pause Continue

Variable

Set background color to Yellow ▼

Monitor

Refresh rate (sec.): 2

Timestamp: 20011023-10:31:59

Status: Loop 1240: Logging Error

Log / Variable

Error 0

Warning 0

Info 0

Debug 0

Set Log Mask

Refresh Totals

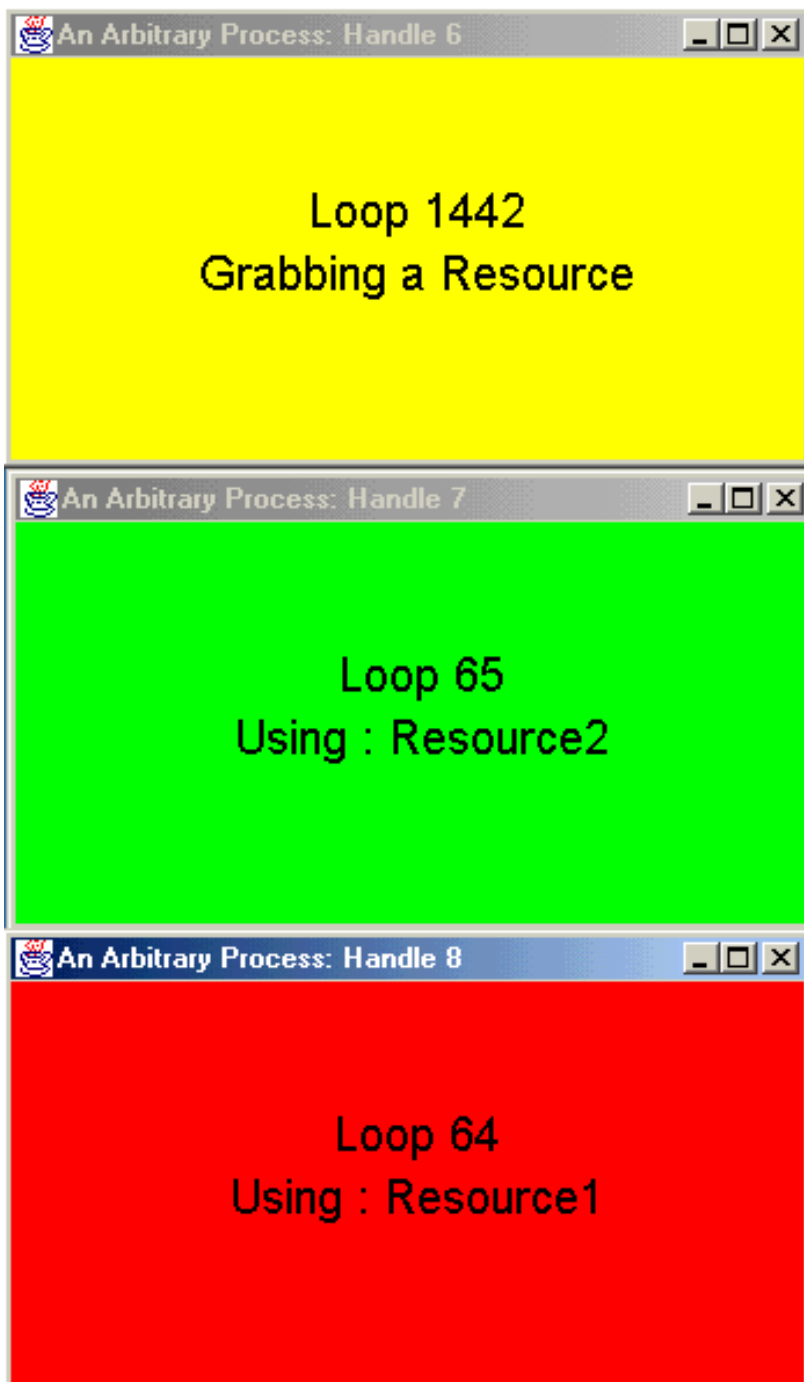
Clear Log

Display Log

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

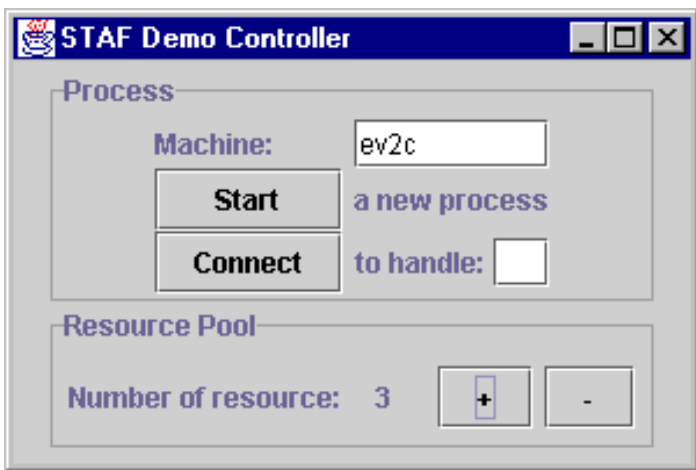
Using the Resource Pool service

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:

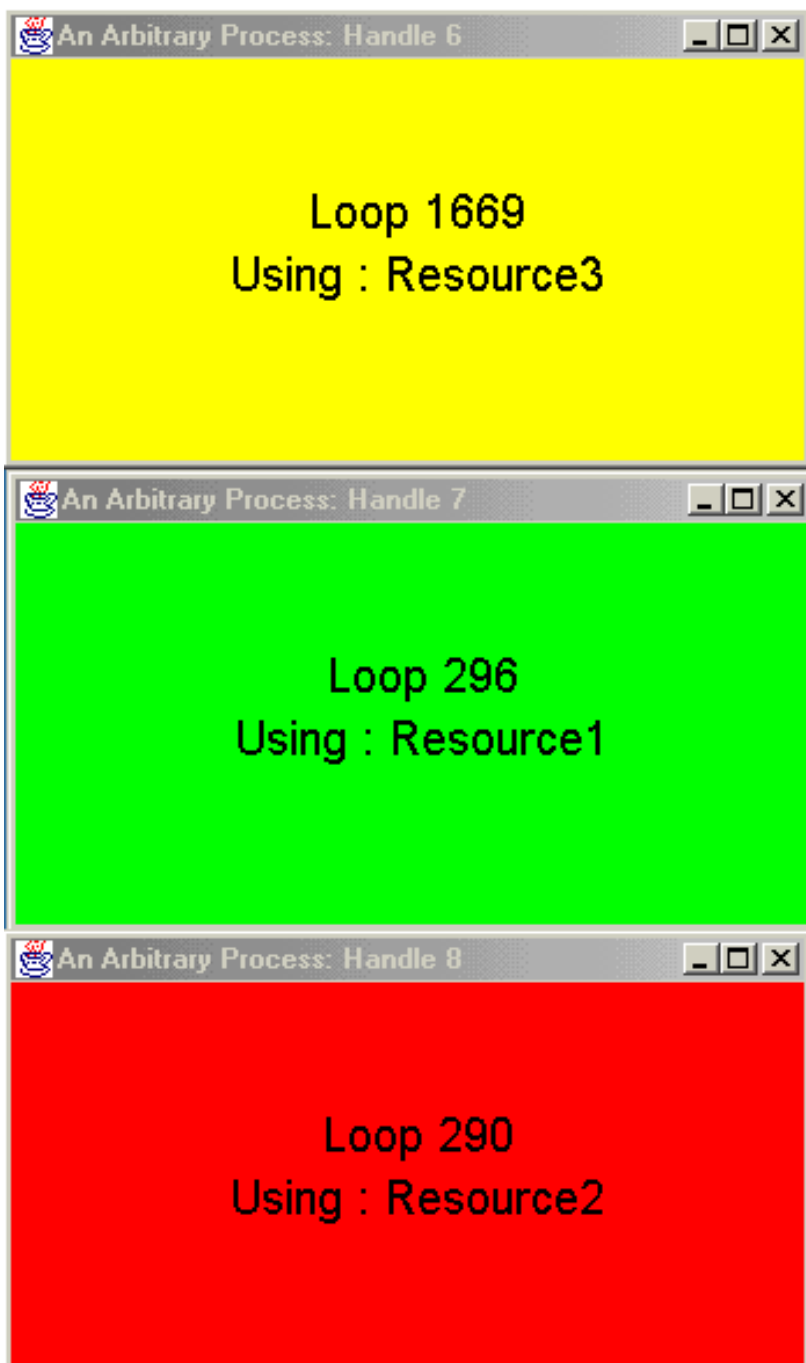


Since there are initially only two resources available, you should notice that only two out of the three applications will be using a resource at any one time; the other application will be waiting for a resource. This is because each application needs a "resource" in order to continue execution. Periodically, 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".

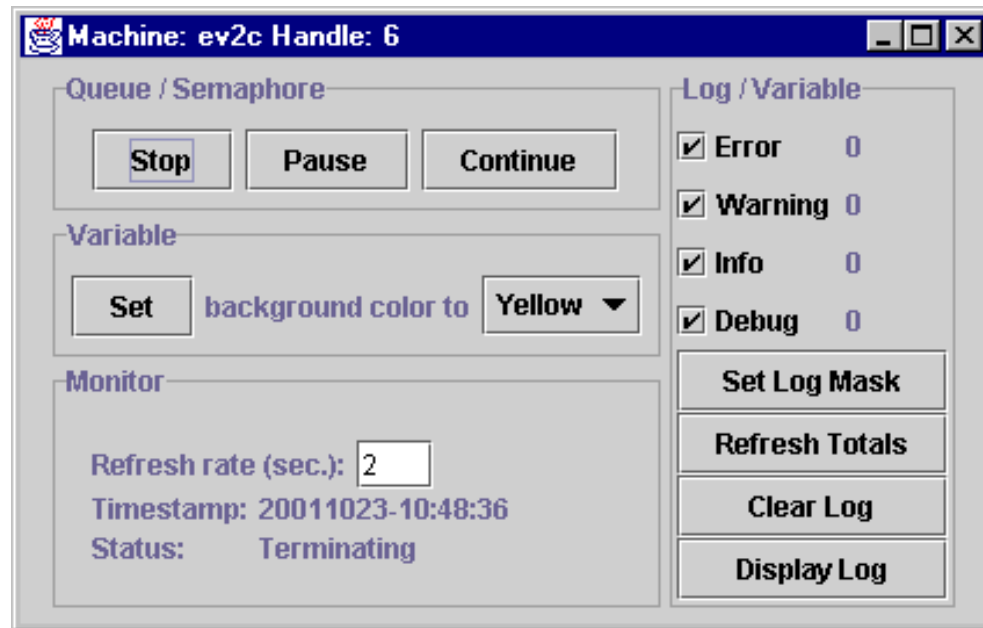


This will add another resource for the applications. You should now see all three applications using resources simultaneously.



Using the Queue service

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



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), 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.

STAF DEMO - Registering and Un-registering with STAF

In this section we'll start looking at the actual Java source code for the STAFDemoController and the 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, 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 7 of STAFDemoController.java shows the import statement.

STAFDemoController.java:

```
1: import java.io.*;
2: import java.util.*;
3: import java.awt.*;
4: import java.awt.event.*;
5: import javax.swing.*;
6: import javax.swing.border.*;
7: 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:

```
21: static STAFHandle handle;
22: static String stafMachineName;
23: static int numResources = 2;
24:
25: public STAFDemoController()
26: {
27:     super("STAF Demo Controller");
28:
29:     try
30:     {
31:         handle = new STAFHandle("STAFDemo_Controller");
32:     }
33:     catch(STAFException e)
34:     {
35:         System.out.println("Error registering with STAF, RC: " + e.rc);
36:         System.exit(e.rc);
37:     }
```

Notice on line 21 that a STAFHandle object is defined. Then on line 31, 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 31 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 STAFDemoController.java, the creation of the STAFHandle is in a try block. On line 33 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 35 and 36 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 81 below).

[STAFDemoController.java](#):

```

74: public void windowClosing(WindowEvent event)
75: {
76:     try
77:     {
78:         handle.submit2("local", "queue", "queue handle " +
79:                       handle.getHandle() +
80:                       " message STAFDemo/Stop");
81:         handle.unregister();
82:     }
83:     catch(STAFException e)
84:     {
85:         System.out.println("Error unregistering with STAF, RC: " +
86:                            e.rc);
87:     }
88:
89:     dispose();
90:     System.exit(0);
91: }
```

STAF DEMO - 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](#):

```

25: public STAFDemoController()
26: {
27:     super("STAF Demo Controller");
28:
29:     try
30:     {
31:         handle = new STAFHandle("STAFDemo_Controller");
32:     }
33:     catch(STAFException e)
34:     {
35:         System.out.println("Error registering with STAF, RC: " + e.rc);
36:         System.exit(e.rc);
37:     }
38:
39:     STAFResult stafResult = handle.submit2("local", "var", "resolve " +
40:         "{STAF/Config/Machine}");
41:
42:     if (stafResult.rc != 0)
43:         System.out.println("Error getting local machine name");
44:
45:     stafMachineName = stafResult.result;

```

On line 39 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 39 that the result of the submit2() call is a STAFResult object. On line 42, the rc variable of this object is examined to determine if the request was successful. On line 45, the result variable of the STAFResult object is accessed to get the String result (the local machine name).

STAF DEMO - 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](#):

```

209: fStart.addActionListener(new ActionListener() {
210:     public void actionPerformed(ActionEvent event)
211:     {

```

```

212:         String machineName = fMachine.getText();
213:
214:         STAFResult stafResult =
215:             handle.submit2(machineName, "PROCESS",
216:                 "start command {STAFDemo/JavaAppCommand} " +
217:                 "parms {STAFDemo/JavaAppParms} " +
218:                 "var STAFDemo/ResourcePoolMachine=" + stafMachineName +
219:                 " var STAF/Service/Log/Mask=Error" +
220:                 " env CLASSPATH={STAFDemo/JavaAppClassPath} " +
221:                 "notify onend");
222:
223:         if (stafResult.rc != 0)
224:             System.out.println("Error starting process");

```

Starting on line 214, there is a call to `submit2` which starts the application on the remote system. Note that the request utilizes the variables that you defined in your `STAF.cfg` file, and it defines some additional variables. Rather than hard-coding these values in `STAFDemoController.java`, we can use STAF's variable substitution to make applications much more portable.

STAF DEMO - 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`:

```

503: fSetBackground.addActionListener(new ActionListener() {
504:     public void actionPerformed(ActionEvent event)
505:     {
506:         STAFResult stafResult =
507:             STAFDemoController.handle.submit2(fMachine, "VAR",
508:                 "handle " + fHandle +
509:                 " set STAFDemo/BackgroundColor=" +
510:                 (String)fColorList.getSelectedItem());
511:
512:         if (stafResult.rc != 0)
513:             System.out.println("Error setting background color");
514:     }
515: });

```

When you click on the "Set" button to change the background color, the `actionPerformed` method at line 504 in `STAFDemoController` is called.

On line 507, 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:

```

70:  public void run()
71:  {
104:      while (true)
105:      {
236:          // sleep for 1 seconds before looping again
237:          handle.submit2(machine, "delay", "delay 1000");
238:          counter++;
239:      }

```

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

STAFProcess.java:

```

90:  String background_color_var = new String("resolve
{STAFDemo/BackgroundColor}");
133: // get the background color, save the old one
134: previous_color = (color == null ? null : new String(color.result));
135: color = handle.submit2(machine, "var", background_color_var);
136:
137: if (color != null && color.rc == STAFResult.Ok)
138: {
139:     // if color changed, log an informational message
140:     if (previous_color != null && !previous_color.equals(color.result))
141:     {
142:         // use monitor as a checkpoint
143:         monitor = handle.submit2(machine, "monitor", "log message \"Loop "+
144:             String.valueOf(counter)+" : Changing Color\");
145:     }
146:
147:     // set background according to color
148:     frame.setBackground(getColorFromString(color.result));
149: }
150: else
151: {
152:     // set background to default (white)
153:     frame.setBackground(Color.white);
154: }

```

On line 135, 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 90.

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.

STAF DEMO - 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`:

```

477:  fPause.addActionListener(new ActionListener() {
478:      public void actionPerformed(ActionEvent event)
479:      {
480:          STAFResult stafResult =
481:              STAFDemoController.handle.submit2(fMachine, "SEM",
482:          "event STAFDemo/Handle/" + fHandle +
483:          "/Continue reset");
484:
485:          if (stafResult.rc != 0)
486:              System.out.println("Error pausing process");
487:      }
488:  });

```

On line 481 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`:

```

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

126: // block if semaphore is reset, fall through if posted (or if error!!!)
127: semaphore = handle.submit2(machine, "sem", continue_semaphore);

```

In `STAFProcess`, on line 127 there is a call to the local Semaphore service. The variable `continue_semaphore` is defined on line 91.

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 application's execution.

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

[STAFDemoController.java](#):

```

490: fContinue.addActionListener(new ActionListener() {
491:     public void actionPerformed(ActionEvent event)
492:     {
493:         STAFResult stafResult =
494:             STAFDemoController.handle.submit2(fMachine, "SEM",
495:         "event STAFDemo/Handle/" + fHandle +
496:         "/Continue post");
497:
498:         if (stafResult.rc != 0)
499:             System.out.println("Error continuing process");
500:     }
501: });

```

When you click on the "Continue" button, line 494 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 127) 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 indicates to the application that it should terminate.

[STAFDemoController.java](#):

```

464: // Add button commands
465: fStop.addActionListener(new ActionListener() {
466:     public void actionPerformed(ActionEvent event)
467:     {
468:         STAFResult stafResult =
469:             STAFDemoController.handle.submit2(fMachine, "QUEUE",
470:         "queue handle " + fHandle + " message STAFDemo/Stop");
471:
472:         if (stafResult.rc != 0)
473:             System.out.println("Error queueing message");
474:     }
475: });

```

When you click on the "Stop" button, line 469 of STAFDemoController will submit a request to the Queue service on the machine where the application is executing. This will result in the message "STAFDemo/Stop" being placed on the queue for the specified handle. Now let's look at how the application processes this message.

[STAFProcess.java:](#)

```

93:   String mesg_queue = new String("get contains STAFDemo/Stop");

113:  // check queue for stop message
114:  stop = handle.submit2(machine, "queue", mesg_queue);
115:
116:  if (stop != null && stop.rc == STAFResult.Ok)
117:  {
118:      // break from the loop
119:      break;
120:  }

```

In STAFProcess' infinite while loop, on line 114 there is a call to the local Queue service. The variable mesg_queue is defined on line 93.

This will check the Queue for the application's handle to determine if there is a message "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.

STAF DEMO - 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:](#)

```

193:  // randomly log an error, warning, debug or information message
194:  switch (counter % 7)
195:  {
196:      // error
197:      case 1:
198:          // use monitor as a checkpoint
199:          monitor = handle.submit2(machine, "monitor", "log message \"Loop "+
200:                                  String.valueOf(counter)+": Logging
Error\");
201:          // do the logging
202:          handle.submit2(machine, "log", mesg_log+"level error "+
203:                          "message \"Loop "+String.valueOf(counter)+": Error
Message\");
204:          break;
205:      // warning
206:      case 2:
207:          // use monitor as a checkpoint
208:          monitor = handle.submit2(machine, "monitor", "log message \"Loop "+

```

```

209:                                     String.valueOf(counter)+" : Logging
Warning\");
210:                                     // do the logging
211:                                     handle.submit2(machine, "log", mesg_log+"level warning "+
212:                                     "message \"Loop "+String.valueOf(counter)+" : Warning
Message\");
213:                                     break;

```

Note that on line 199, a request is submitted to the Monitor service to write an error message. On line 202, a request is submitted to the Log service to log an error message. "case 2" starting on line 206 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](#):

```

726: STAFResult stafResult =
727:     STAFDemoController.handle.submit2(fMachine, "MONITOR",
728:     "query machine " + fMachine + " handle " + fHandle);
729:
730: if (stafResult.rc != 0)
731: {
732:     fDateField.setText("Unknown");
733:     fStatusField.setText("Unknown");
734: }
735: else
736: {
737:     int firstSpace = stafResult.result.indexOf(' ');
738:     String dateText =
739:         stafResult.result.substring(0, firstSpace);
740:     String messageText =
741:         stafResult.result.substring(firstSpace + 1);
742:     fDateField.setText(dateText);
743:     fStatusField.setText(messageText);
744: }

```

STAFDemoController has a separate thread that periodically queries the application's Monitor. This call is on line 727. The Timestamp and Status information is updated based on the request result.

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

[STAFDemoController.java](#):

```

596: fDisplayLog.addActionListener(new ActionListener() {
597:     public void actionPerformed(ActionEvent event)
598:     {
599:         STAFResult stafResult =
600:             STAFDemoController.handle.submit2(fMachine, "LOG",
601:             "query machine " + fMachine + " handle " + fHandle +

```

```

602:         " logname STAFDemo");
603:
604:         if ((stafResult.rc != 0) && (stafResult.rc != 17))
605:             System.out.println("Error querying log");
606:
607:         BufferedReader logFile = new BufferedReader(
608:             new StringReader(stafResult.result));

```

When you click on the "Display Log" button, on line 600 of STAFDemoController, a request will be submitted to the Log service to query the log information. The result is then displayed in the table.

STAF DEMO - Using the Resource Pool Service

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

[STAFDemoController.java](#):

```

52:  stafResult = handle.submit2("local", "respool", "create pool " +
53:      "STAFDemo description \"" +
54:      "STAF Demo Resource Pool\"");
55:  if (stafResult.rc != 0)
56:      System.out.println("Error creating STAFDemo resource pool");
57:
58:  for(int i = 1; i <= numResources; ++i)
59:  {
60:      stafResult = handle.submit2("local", "respool", "add pool " +
61:          "STAFDemo entry Resource" + String.valueOf(i));
62:      if (stafResult.rc != 0)
63:          System.out.println("Error adding resource to  STAFDemo " +
64:              "resource pool");
65:  }

```

In STAFDemoController, on line 52 a Resource Pool titled "STAFDemo" is created. Then, on line 60, 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](#):

```

92:  String items_respool = new String("request pool STAFDemo random");

177:  // block until resource is available
178:  resource = handle.submit2("{STAFDemo/ResourcePoolMachine}", "respool",
179:      items_respool);

```

On line 178 of STAFProcess, a request is submitted to the ResPool service to request a resource. The

variable items_respool is defined on line 92. 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.

GLOSSARY

external services - Services for which the executable code for external STAF services 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.

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 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.

workload - A set of processes running on a set of machines.