Event Service User's Guide

Version 3.1.4

Last updated: December 1, 2009

Table of Contents

Overview

The EVENT service is an external STAF service. The purpose of the EVENT service is to provide an interface to allow process communication based on events occuring. For instance, a process registers with the EVENT service that it is interested in builds completing. A different process tell the EVENT service when a build has completed. The EVENT service then notifies all processes that have registered interest in the build completing that the build has completed.

Communication is done by using the STAF QUEUE service. The EVENT service notifies a process that an event has occurred by queueing a message with queue type "STAF/Service/Event" with the content of the message being a `<Map>` defined as follows:

Definition for message map for a "STAF/Service/Event" type queued message

Description: This map represents information about a generated event.

Key Name	Type	Format / Value
eventServiceName	`<String>`	
eventID	`<String>`	
machine	`<String>`	
handleName	`<String>`	
handle	`<String>`	
timestamp	`<String>`	`<YYYYMMDD-HH:MM:SS>`
type	`<String>`	
subtype	`<String>`	
propertyMap	`<Map>`	

Notes:

1. Keys "machine", "handleName", and "handle" represent the machine, handle name, and handle number of the process who generated the event.
2. The value for "propertyMap" will contain a map representing the specified properties. If no `PROPERTY` options were specified on the generate request, the map will be empty. Otherwise, for each `PROPERTY <Name>=[<Value>]` specified, the property map will contain an entry with key set to `<Name>` and it's value set to `<Value>` (or `<None>` if no `<Value>` is specified).

Here's an example of a queued message with type "STAF/Service/Event" generated by the STAX service for use by the STAX Monitor:

```
STAF local QUEUE GET WAIT

{
  Priority   : 5,
  Date-Time  : 20041019-21:52:45
  Machine    : mach1
  Handle Name: STAF/SERVICE/Event
  Handle     : 8
  User       : none://anonymous
  Type       : STAF/Service/Event
  Message    : {
    eventID         : 5686
    eventServiceName: Event
    handle          : 36
    handleName      : STAX/Job/1
    machine         : mach1.austin.ibm.com
    propertyMap     : {
      elapsed-time: 00:00:34
      laststatus  : pass
      message     :
      name        : TestSTAF.Win2000.local.DEVICE_C++
      num-starts  : 68
      status      : update
      status-fail : 0
      status-pass : 68
    }
    subtype         : TestcaseStatus
    timestamp       : 20041019-21:52:45
    type            : STAX/mach1/1
  }
}
```

Installation and Configuration

1. Install Java 1.2 or later.

2. Install STAF 3.1.0 or later by following the installation instructions in the STAF documentation.

3. Install the Event V3.1.4 service:

   Download the EventV314.zip/tar file from Get STAF Services into a local directory (e.g. C:/STAF/services or /usr/local/staf/services) and extract it.

4. Configure the Event service:

   Add the following statement to your staf.cfg file:

   ```
   SERVICE <Service Name> LIBRARY JSTAF EXECUTE <Service Jar File Name>
         [PARMS "MAXATTEMPTS <Max Attempts> ACKNOWLEDGETIMEOUT <Timeout>"]
   ```

   where:

   - `<Service Name>` is the name by which the Event service will be known on this machine.
   - `<Service Jar File Name>` is the fully-qualified name of the STAFEvent.jar file.
   - `<Max Attempts>` is the maximum number of times that the EVENT Service should notify this process that the event has occurred. The default is 1. This option will resolve STAF variables.
   - `<Timeout>` is the length of time between notification attempts in milliseconds. The default is 60000 (60 seconds). This option will resolve variables.

     Examples:

     ```
     SERVICE Event LIBRARY JSTAF EXECUTE {STAF/Config/STAFRoot}/services/event/
     STAFEvent.jar

     SERVICE Event LIBRARY JSTAF EXECUTE C:/STAF/services/event/STAFEvent.jar
     ```

```
SERVICE Event LIBRARY JSTAF EXECUTE /usr/local/staf/services/event/
STAFEvent.jar

SERVICE Event LIBRARY JSTAF EXECUTE C:/STAF/services/event/STAFEvent.jar \
              PARMS "MAXATTEMPTS 2 ACKNOWLEDGETIMEOUT 80000"
```

Or, you can dynamically add the EVENT service using the SERVICE service's ADD SERVICE request.

In most cases, the way that you would use this service is that you would designate a system as the EVENT Server. That system would have STAF installed, plus the EVENT service with the SERVICE configuration statement in the configuration file that is shown above. Assuming that the machine name for the Event server is EventServer, an example of the command line interface to generate an event would be:

```
STAF EventServer EVENT GENERATE TYPE MyEvent SUBTYPE SubtypeA
```

Request Syntax

The Event service provides the following requests:

- REGISTER- registers with the EVENT service to be notified of an event
- UNREGISTER - unregister for a previously registered event
- ACKNOWLEDGE - tell the EVENT service that an event has been recieved
- GENERATE - tell the EVENT service that an event has occurred so that the service can tell all processes REGISTERed for that event that the event has occurred
- LIST - returns a list of types, subtypes, registrations, event ids, or service operational settings depending on the options selected.
- QUERY - returns information about the specified event ID including its event type, subtype, and properties, who generated the event, and the pending notifications for this event.
- RESET - clears all registered and/or generated events
- VERSION - displays the version of the running event service
- HELP - provides a list of commands and how to use them

REGISTER

REGISTER tells the EVENT service that the requesting process is interested in a particular event. If a process needs to be notified about multiple types of events, it can use multiple invocations of the REGISTER command.

Syntax

```
REGISTER TYPE <Type> [SUBTYPE <Subtype>]...
         [ACKNOWLEDGETIMEOUT <Timeout>] [MAXATTEMPTS <Number>]
         [PRIORITY <Number>] [PRIORITYDELTA <Number>]
         [BYNAME | BYHANDLE]
```

`TYPE` specifies the type of event. This can be any string, but must match the `TYPE` value specified on the `GENERATE` request in order to be notified for this event. Note that this match is case insensitive. This option will resolve variables.

`SUBTYPE` is used to further qualify this event. Any number of `SUBTYPE` options may be specified. One of the `<value>`'s specified for a `SUBTYPE` must match the (unique) `SUBTYPE` value specified on the `GENERATE` request in order to be notified for this event. Note that this match is case insensitive. If no `SUBTYPE`s were specified on the `REGISTER` command, the process will be notified for all `SUBTYPE`s of a given event `TYPE`. This option will resolve variables.

`ACKNOWLEDGETIMEOUT` specifies the length of time in milliseconds between notification attempts. If not specified, the default timeout specified when configuring the Event service is used. This option will resolve variables.

`MAXATTEMPTS` specifies the maximum number of times that the EVENT Service should notify this process that the event has occurred. If not specified, the default maximum attempts specified when configuring the Event service is used. This option will resolve variables.

`PRIORITY` specifies the priority of the event notification. The default is 5. This option will resolve variables.

`PRIORITYDELTA` specifies a number that will be used to decrement the priority for this process after the process is notified of the event. The default is 1. This option will resolve variables.

`BYNAME` indicates that the process should be registered by handle name.

`BYHANDLE` indicates that the process should be registered by handle number rather than handle name. If the `BYHANDLE` option is not selected, registration will be by handle name.

Security

This command requires trust level 3.

Results

Upon successful return, the result buffer will be empty.

Examples:

- **Goal:** Register the requesting process by handle name for events with type build:

  ```
  REGISTER TYPE build
  ```

- **Goal:** Register the requesting process by handle name for events with type build, but only for subtype Windows:

  ```
  REGISTER TYPE build SUBTYPE Windows
  ```

- **Goal:** Register the requesting process by handle name for events with type build, but only for subtypes Linux, AIX, and HP-UX:

  ```
  REGISTER TYPE build SUBTYPE Linux SUBTYPE AIX SUBTYPE HP-UX
  ```

- **Goal:** Register the requesting process by handle number rather than by handle name for events with type build:

  ```
  REGISTER TYPE build BYHANDLE
  ```

- **Goal:** Register the requesting process by handle name for events with type build, making 2 notification attempts (rather the the default of 1 attempt) with a 30 second timeout between notification attempts:

  ```
  REGISTER TYPE build MAXATTEMPTS 2 ACKNOWLEDGETIMEOUT 30000
  ```

- **Goal:** Register the requesting process by handle number for events with type build, but only for subtypes En and Sp. Make a maximum of 3 notification attempts if not acknowledgement is received:

  ```
  REGISTER TYPE build SUBTYPE En SUBTYPE Sp MAXATTEMPTS 3 BYHANDLE
  ```

ACKNOWLEDGE

ACKNOWLEDGE is used to tell the EVENT Service that an event has been received.

Syntax

```
ACKNOWLEDGE EVENTID <Event ID>
            [FORCE [MACHINE <Machine>] [HANDLE <Handle> | NAME <Name>]]
```

`EVENTID` specifies the Event ID that was part of the message queued to the process when the process was notified that the event occurred.

`FORCE` allows one process to acknowledge an event for a different process.

`MACHINE` is used with `FORCE` to force acknowledgement for a process running on `MACHINE <Machine>`.

`HANDLE` is used with `FORCE` to force acknowledgement for a process with `HANDLE <Handle>`.

`NAME` is used with `FORCE` to force acknowledgment for a process with handle `NAME <Name>`.

Security

This command requires trust level 3 unless the `FORCE` option is specified which requires trust level 4.

Results

Upon successful return, the result buffer will contain the Event ID.

Examples

- **Goal:** Notify Event Service that process "MyProcess" has successfully retrieved message for Event ID 25 from the "MyProcess" STAF queue:

  ```
  ACKNOWLEDGE EVENTID 25
  ```

- **Goal:** Acknowledge that process "YourProcess" on machine machineA has been notified for Event ID 25 from "SomeProcess":

  ```
  ACKNOWLEDGE EVENTID 25 FORCE MACHINE machineA NAME YourProcess
  ```

- **Goal:** Acknowledge that the process with handle 7 on machine machineA has been notified for Event ID 25.

  ```
  ACKNOWLEDGE EVENTID 25 FORCE MACHINE machineA HANDLE 7
  ```

- **Goal:** Acknowledge that process "YourProcess" on the local machine has been notified for Event ID 25.

  ```
  ACKNOWLEDGE EVENTID 25 FORCE NAME YourProcess
  ```
  or
  ```
  ACKNOWLEDGE EVENTID 25 FORCE MACHINE local NAME YourProcess
  ```

GENERATE

Generate notifies the Event Service that an event has occurred.

Syntax

```
GENERATE TYPE <Type> SUBTYPE <Subtype> [PROPERTY <Name>[=<Value>]]...
```

`TYPE` specifies the type of event. This can be any string, but must match the `TYPE` value specified on the `REGISTER` request in order to be notified for this event. Note that this match is case insensitive. This option resolves variables.

`SUBTYPE` is used to further qualify this event. This can be any string, but must match the `SUBTYPE` value specified on the `REGISTER` command in order to be notified for this event. Note that this match is case insensitive. This option resolves variables.

`PROPERTY` is used to pass information to further qualify the event. The format is `<Name>[=<Value>]`. This option does not resolve variables. This option will handle private data in the <Value> specified for a property.

Security

This command requires trust level 4.

Results

Upon successful return, the result buffer will contain the Event ID of the generating event.

Examples

- **Goal:** Notify the Event Service that an event of type "build" and subtype "Sp" has occurred:

  ```
  GENERATE TYPE build SUBTYPE Sp
  ```

- **Goal:** Notify the Event Service that an event of type "build" and subtype "Sp" with the following three properties: Version=2.1.0, "Status=Passed BVT", and Save:

  ```
  GENERATE TYPE build SUBTYPE Sp PROPERTY Version=2.1.0 PROPERTY "Status=Passed
  BVT" PROPERTY Save
  ```

UNREGISTER

UNREGISTER tells the Event Service to no longer notify this process about events of specified TYPE and SUBTYPE for which this process was previously registered.

Syntax

```
UNREGISTER TYPE <Type> [SUBTYPE <Subtype>]...
           [FORCE [MACHINE <Machine>] [HANDLE <Handle> | NAME <Name>]]
```

TYPE specifies the type of event. This can be any value, but must match the TYPE value specified on the REGISTER request. Note that this match is case insensitive. This option resolves variables.

SUBTYPE is used to further qualify this event. This can be any value, but must match the SUBTYPE value specified on the REGISTER request in order to unregister the process for that SUBTYPE. Note that this match is case insensitive. This option resolves variables.

FORCE allows one process to unregister a different process from the submitting process.

MACHINE is used with FORCE to force unregistration of a process running on MACHINE <Machine>.

HANDLE is used with FORCE to force unregistration of a process with HANDLE <Handle>.

NAME is used with FORCE to force unregistration of a process with handle NAME <Name>.

Security

This command requires trust level 3, unless the FORCE option is specified which requires trust level 4.

Results

Upon successful return, the result buffer will be empty.

Examples

- **Goal:** Unregister the requesting process by handle name, the default, for all events of TYPE build:

  ```
  UNREGISTER TYPE build
  ```

- **Goal:** Unregister the process(es) with handle name "YourProcess" on machine machineA for all events of TYPE build.

  ```
  UNREGISTER TYPE build FORCE MACHINE machineA NAME YourProcess
  ```

- **Goal:** Unregister the process with handle 7 on machine machineA for events of TYPE build, SUBTYPEs sp and fr only.

  ```
  UNREGISTER TYPE build SUBTYPE sp SUBTYPE fr FORCE MACHINE machineA HANDLE 7
  ```

- **Goal:** Unregister the process(es) with handle name "YourProcess" on the local machine for events of TYPE build..

```
UNREGISTER TYPE build FORCE NAME YourProcess
```
or
```
UNREGISTER TYPE build FORCE MACHINE local NAME YourProcess
```

LIST

LIST returns a list of types, subtypes, registrations, event ids, or service operational settings depending on the options selected.

Syntax

```
LIST TYPES [LONG]
LIST SUBTYPES TYPE <Type>
LIST REGISTRATIONS [TYPE <Type> [SUBTYPE <Subtype>]] [LONG]
LIST EVENTIDS [LONG]
LIST SETTINGS
```

`TYPES` specifies to list the event types that are registered.

`SUBTYPES` specifies to list the event subtypes that are registered for the specified event type.

`REGISTRATIONS` specifies to list the processes that are registered for events as follows:

- If `TYPE` is not specified, all of the registered processes along with the `TYPE`s and `SUBTYPE`s for which they are registered are listed.
- If `TYPE` is specified, but not `SUBTYPE`, all of the processes registered for events of the specified type are listed, regardless of the subtype.
- If `TYPE` and `SUBTYPE` are specified, all of the processes registered for events of the specified type and subtype are listed.

`TYPE` specifies the type of event. This can be any value, but must match the `TYPE` value specified on the `REGISTER` request. Note that this match is case insensitive. This option resolves variables.

`SUBTYPE` is used to further qualify this event. This can be any value, but must match the `SUBTYPE` value specified on the `REGISTER` request. Note that this match is case insensitive. This option resolves variables.

`EVENTIDS` specifies to list information about all Event IDs.

`LONG` specifies to list additional information about the type, registrations, or event IDs.

`SETTINGS` is specified, the operational settings for the Event service are listed. The operational settings include the default values for maximum attempts, acknowledge timeout, priority, and priority delta.

Security

This command requires trust level 2.

Results

Upon successful return,

- The result buffer for a `"LIST TYPES"` request will contain a marshalled `<List> of <String>` which represents a list of all registered event types.

- The result buffer for a `"LIST TYPES LONG"` request will contain a marshalled `<List> of <Map:STAF/Service/Event/Type>` which represents a list of all registered event types and their subtypes. The map is defined as follows:

Definition of map class STAF/Service/Event/Type			
Description: This map class represents an event type and its subtypes.			
Key Name	**Display Name**	**Type**	**Format / Value**
type	Type	`<String>`	
subtypeList	Subtypes	`<List> of <String>`	

- The result buffer for a `"LIST SUBTYPES TYPE <Type>"` request will contain a marshalled `<List> of <String>` which represents a list of all registered subtypes for the specified event type.

- The result buffer for a `LIST REGISTRATIONS` request (without the LONG option) will contain a marshalled `<List> of <Map:STAF/Service/Event/ListRegistrations>` which represents a list of registrations for events. The map is defined as follows:

Definition of map class STAF/Service/Event/ListRegistrations			
Description: This map class represents a registration for an event.			
Key Name	**Display Name**	**Type**	**Format / Value**
type	Type	`<String>`	
subtype	Subtype	`<String> | <None>`	
machine	Machine	`<String>`	
notifyBy	Notify By	`<String>`	`'Name' | 'Handle'`
notifiee	Notifiee	`<String>`	
Notes: 1. If "Subtype" is <None>, this indicates that a process did not specify a specific subtype when registering so that any subtype for an event with the specified type will match. 2. "Machine" is the machine which registered to be notified for this event. 3. "Notify By" indicates whether the process registered by handle name ('Name') or by handle number ('Handle'). 4. "Notifiee" is the handle name or handle number of the process that registered to be notified of this event.			

- The result buffer for a `LIST REGISTRATIONS LONG` request will contain a marshalled `<List> of <Map:STAF/Service/Event/ListRegistrationsLong>` which represents a list of registrations for events. The map is defined as follows:

Definition of map class STAF/Service/Event/ListRegistrationsLong
Description: This map class represents a registration for an event, with detailed information.

Key Name	Display Name	Type	Format / Value
type	Type	`<String>`	
subtype	Subtype	`<String> | <None>`	
machine	Machine	`<String>`	
notifyBy	Notify By	`<String>`	`'Name' | 'Handle'`
notifiee	Notifiee	`<String>`	
attempts	Max Attempts (Att)	`<String>`	
timeout	Timeout	`<String>`	
priority	Priority (P)	`<String>`	
priorityDelta	Priority Delta (D)	`<String>`	

Notes:

1. If "Subtype" is <None>, this indicates that a process did not specify a specific subtype when registering so that any subtype for an event with the specified type will match.
2. "Machine" is the machine which registered to be notified for this event.
3. "Notify By" indicates whether the process registered by handle name ('Name') or by handle number ('Handle').
4. "Notifiee" is the handle name or handle number of the process that registered to be notified of this event.
5. "Max Attempts" is the maximum number of attempts that will be made to notify a registered process that this event occurred, specified when registering the type/subtype (or the default if not specified).
6. "Timeout" is the acknowledge timeout, the length of time in milliseconds between notification attempts, specified when registering the type/subtype (or the default if not specified).
7. "Priority" is the starting priority specified when registering the type/subtype (or the default if not specified).
8. "Priority Delta" is a number that will be used to decrement the priority after each notification attempt, specified when registering the type/subtype (or the default if not specified).

- The result buffer for a `LIST EVENTIDS` request (without the LONG option) will contain a marshalled `<List> of <Map:STAF/Service/Event/EventID>` which represents a list of event IDs with at least one pending notification. The maps is defined as follows:

Definition of map class STAF/Service/Event/EventID

Description: This map class represents an event ID that has at least one pending notification.

Key Name	Display Name	Type	Format / Value
eventID	Event ID	`<String>`	
type	Type	`<String>`	
subtype	Subtype	`<String>`	
numNotifiees	# Notifiees	`<String>`	

Notes: The "# Notifiees" value is set to the number of registered processes that have been notified, but have not yet acknowledged, the event ID. There will be at least 1 notifiee.

- The result buffer for a `LIST EVENTIDS LONG` request will contain a marshalled `<List> of <Map:STAF/Service/Event/EventIDLong>` which represents a more detailed of event IDs with at least one pending notification. See table "Definition of map class STAF/Service/Event/EventIDLong" for a description of this map.

- The result buffer for a `LIST SETTINGS` request will contain a marshalled `<Map:STAF/Service/Event/Settings>` which represents a list of the operational settings for the Event service. The map is defined as follows:

Definition of map class STAF/Service/Event/Settings

Description: This map class represents the operational settings for the service.

Key Name	Display Name	Type	Format / Value
maxAttempts	Maximum Attempts	`<String>`	
ackTimeout	Acknowledge Timeout	`<String>`	
priority	Priority	`<String>`	
priorityDelta	Priority Delta	`<String>`	

Examples

For the following examples, assume that there are only 2 processes registered, both for TYPE build (SpanishJavaTest and EnglishJavaTest). SpanishJavaTest is registered only for SUBTYPE Sp. EnglishJavaTest is not registered for any specific SUBTYPE's. Both processes are runing on machine client2.austin.ibm.com. All option values are the default REGISTRATION values.

- **Goal:** List all types:

  ```
  LIST TYPES
  ```

  Result: If the request is submitted from the command line, the result, in the default format, would look like:

  ```
  build
  ```

- **Goal:** List all event types, along with their subtypes:

  ```
  LIST TYPES
  ```

  Result: If the request is submitted from the command line, the result, in the verbose format, would look like:

  ```
  [
    {
      Type    : build
      Subtypes: [
        sp
      ]
    }
  ]
  ```

- **Goal:** List all subtypes of type build:

  ```
  LIST SUBTYPES TYPE build
  ```

Result: If the request is submitted from the command line, the result, in the default format, would look like:

```
Sp
```

- **Goal:** List all registered processes (in the short format):

  ```
  LIST REGISTRATIONS
  ```

  Result:

  ```
  Type  Subtype Machine               Notify By Notifiee
  ----- ------- --------------------- --------- ---------------
  build Sp      client2.austin.ibm.com Name      SpanishJavaTest
  build <None>  client2.austin.ibm.com Name      EnglishJavaTest
  ```

- **Goal:** List all registered processes (in the long format):

  ```
  LIST REGISTRATIONS LONG
  ```

  Result: If the request is submitted from the command line, the result, in the verbose format, would look like:

  ```
  [
    {
      Type          : build
      Subtype       : Sp
      Machine       : client2.austin.ibm.com
      Notify By     : Name
      Notifiee      : SpanishJavaTest
      Max Attempts  : 1
      Timeout       : 60000
      Priority      : 5
      Priority Delta: 1
    }
    {
      Type          : build
      Subtype       : <None>
      Machine       : client2.austin.ibm.com
      Notify By     : Name
      Notifiee      : EnglishJavaTest
      Max Attempts  : 1
      Timeout       : 60000
      Priority      : 5
      Priority Delta: 1
    }
  ]
  ```

- **Goal:** List processes registered to be notified of event type build (in the short format):

  ```
  LIST REGISTRATIONS TYPE build
  ```

Result: If the request is submitted from the command line, the result, in the table format, would look like:

```
Type  Subtype Machine                Notify By Notifiee
----- ------- ---------------------- --------- ---------------
build Sp      client2.austin.ibm.com Name      SpanishJavaTest
build <None>  client2.austin.ibm.com Name      EnglishJavaTest
```

- **Goal:** List processes registered to be notified of event type build and subtype Sp (in the long format):

  ```
  LIST REGISTRATIONS TYPE build SUBTYPE Sp LONG
  ```

  Result: If the request is submitted from the command line, the result, in the verbose format, would look like:

  ```
  [
    {
      Type          : build
      Subtype       : Sp
      Machine       : client2.austin.ibm.com
      Notify By     : Name
      Notifiee      : SpanishJavaTest
      Max Attempts  : 1
      Timeout       : 60000
      Priority      : 5
      Priority Delta: 1
    }
  ]
  ```

- **Goal:** List all processes that have been notified about any Event ID but have not yet acknowledged (assume 2 events of TYPE build have generated, and one event of TYPE build and SUBTYPE en, generated from the STAF command line with generating handle 30, and event ID's 310 and 311, respectively):

  List in the short format:

  ```
  LIST EVENTIDS
  ```

  Result: If the request is submitted from the command line, the result, in the table format, would look like:

  ```
  Event ID Type  Subtype # Notifiees
  -------- ----- ------- -----------
  310      build Sp      2
  311      build En      1
  ```

  List in the long format:

  ```
  LIST EVENTIDS LONG
  ```

  Result: If the request is submitted from the command line, the result, in the verbose format, would look like:

  ```
  [
    {
  ```

```
        Event ID             : 310
        Type                 : build
        Subtype              : Sp
        Properties           : {
          Status : Passed BVT
          Version: 2.1.0
        }
        Generated By         : {
          Machine    : client2.austin.ibm.com
          Handle Name: STAF/Client
          Handle     : 30
        }
        Pending Notifications': [
          {
            Machine            : client2.austin.ibm.com
            Notify By          : Name
            Notifiee           : SpanishJavaTest
            Attempts Remaining: 0
            Timeout            : 60000
            Priority           : 4
            Priority Delta     : 1
          }
          {
            Machine            : client2.austin.ibm.com
            Notify By          : Name
            Notifiee           : EnglishJavaTest
            Attempts Remaining: 0
            Timeout            : 60000
            Priority           : 4
            Priority Delta     : 1
          }
        ]
      }
      {
        Event ID             : 311
        Type                 : build
        Subtype              : En
        Properties           : {
          Status : Failed BVT
          Version: 2.1.0
        }
        Generated By         : {
          Machine    : client2.austin.ibm.com
          Handle Name: STAF/Client
          Handle     : 30
        }
        Pending Notifications: [
          {
            Machine            : client2.austin.ibm.com
            Notify By          : Name
            Notifiee           : EnglishJavaTest
            Attempts Remaining: 0
            Timeout            : 60000
            Priority           : 4
            Priority Delta     : 1
          }
        ]
      }
    ]
```

- **Goal:** List the operational settings for the event service:

  ```
  LIST SETTINGS
  ```

  Result: If the request is submitted from the command line, the result could look like:

  ```
  Maximum Attempts       : 1
  Acknowledgement Timeout: 60000
  Priority               : 5
  Priority Delta         : 1
  ```

QUERY

QUERY returns information about the specified event ID including its event type, subtype, and properties, who generated the event, and the pending notifications for this event. A pending notification is one where a registered process was been notified but has not yet acknowledged the event ID. Only event IDs with one or more pending notifications can be queried.

Syntax

```
QUERY EVENTID <Event ID> [LONG]
```

`EVENTID` specifies the event ID to query.

`LONG` specifies to provide more information about the pending notifications for the specified event ID instead of just the number of pending notifications.

Security

This command requires trust level 2.

Results

Upon successful return:

- The result buffer for a `"QUERY EVENTID <Event ID>"` request will contain a marshalled `<Map:STAF/Service/Event/QueryEventID>` which represents an event ID that has at least one pending notification. The maps are defined as follows:

Definition of map class STAF/Service/Event/QueryEventID
Description: This map class represents an event ID that has at least one pending notification.

Key Name	Display Name	Type	Format / Value
eventID	Event ID	`<String>`	
type	Type	`<String>`	
subtype	Subtype	`<String>`	
propertyMap	Properties	`<Map>`	Private data will be masked.
generatedBy	Generated By	`<Map:STAF/Service/Event/Generator>`	
numNotifiees	# Notifiees	`<String>`	

Notes:
1. Each entry in the "Properties" map will have a key set to the specified property name, and the value will be set to the specified value for the property (or <None> if no value is specified).
2. The "# Notifiees" value is set to the number of registered processes that have been notified, but have not yet acknowledged, the event ID. There will be at least 1 notifiee.

Definition of map class STAF/Service/Event/Generator

Description: This map class represents who generated the event ID.

Key Name	Display Name	Type	Format / Value
machine	Machine	`<String>`	
handleName	Handle Name	`<String>`	
handle	Handle	`<String>`	

Notes:
1. "Machine" is the machine where the event ID was generated.
2. "Handle Name" is the handle name of the process that generated the event ID.
3. "Handle" is the handle number of the process that generated the event ID.

- The result buffer for a `"QUERY EVENTID <Event ID> LONG"` request will contain a marshalled `<Map:STAF/Service/Event/EventIDLong>` which represents an event ID that has at least one pending notification. This map class provides additional information about the pending notifications for the event ID. The maps are defined as follows:

Definition of map class STAF/Service/Event/EventIDLong

Description: This map class represents an event ID that has at least one pending notification and provided more detailed information about the event ID.

Key Name	Display Name	Type	Format / Value
eventID	Event ID	`<String>`	
type	Type	`<String>`	
subtype	Subtype	`<String>`	
propertyMap	Properties	`<Map>`	Private data will be masked.
generatedBy	Generated By	`<Map:STAF/Service/Event/Generator>`	

notificationList	Pending Notifications	`<List> of <Map:STAF/Service/Event/PendingNotification>`

Notes:

1. Each entry in the "Properties" map will have a key set to the specified property name, and the value will be set to the specified value for the property (or <None> if no value is specified).
2. See table "Definition of map class STAF/Service/Event/Generator" for a description of this map.
3. Each entry in the "Pending Notifications" list contains information about a registered process that has been notified, but have not yet acknowledged, the event ID. There will be at least 1 notifiee in the list.

Definition of map class STAF/Service/Event/PendingNotification

Description: This map class represents a pending notification for the event ID.

Key Name	Display Name	Type	Format / Value
machine	Machine	`<String>`	
notifyBy	Notify By	`<String>`	`'Name' | 'Handle'`
notifiee	Notifiee	`<String>`	
attempts	Attempts Remaining	`<String>`	
timeout	Timeout	`<String>`	
priority	Priority	`<String>`	
priorityDelta	Priority Delta	`<String>`	

Notes:

1. "Machine" is the machine which registered to be notified for this event.
2. "Notify By" indicates whether the process registered by handle name ('Name') or by handle number ('Handle').
3. "Notifiee" is the handle name or handle number of the process that registered to be notified of this event.
4. "Attempts Remaining" is the number of attempts remaining to notify the registered process that this event occurred.
5. "Timeout" is the acknowledge timeout which is the length of time in milliseconds between notification attempts.
6. "Priority" is the current priority.
7. "Priority Delta" is a number that is used to decrement the priority after each notification attempt.

Examples

For the following examples, assume that there are 2 processes (SpanishFVTTest and SpanishSVTTest) registered for events with `TYPE` build and `SUBTYPE` Sp. Process SpanishFVTTest is running on machine client2.austin.ibm.com and process SpanishSVTTest is running on machine client3.austin.ibm.com. All option values are the default registration values.

- **Goal:** Query processes that have been notified about Event ID 310 (of `TYPE` build, `SUBTYPE` sp, `PROPERTY` Version=2.1.0, and `PROPERTY` "Status=Passed BVT", generated from the STAF command line on machine client1.austin.ibm.com by handle 25), but which have not yet been acknowledged.

  ```
  QUERY EVENTID 310
  ```

  Result: If the request is submitted from the command line, the result, in the verbose format, would look like:

  ```
  (
  ```

```
    Event ID    : 310
    Type        : build
    Subtype     : Sp
    Properties  : {
      Status : Passed BVT
      Version: 2.1.0
    }
    Generated By: {
      Machine    : client1.austin.ibm.com
      Handle Name: STAF/Client
      Handle     : 25
    }
    # Notifiees : 2
  }
```

- **Goal:** Query processes that have been notified about Event ID 310 (of `TYPE` build, `SUBTYPE` sp, `PROPERTY` Version=2.1.0, and `PROPERTY` "Status=Passed BVT", generated from the STAF command line on machine client1.austin.ibm.com by handle 25), but which have not yet been acknowledged. Use the `LONG` format option:

  ```
  QUERY EVENTID 310 LONG
  ```

  Result: If the request is submitted from the command line, the result, in the verbose format, would look like:

  ```
  {
    Event ID             : 310
    Type                 : build
    Subtype              : Sp
    Properties           : {
      Status : Passed BVT
      Version: 2.1.0
    }
    Generated By         : {
      Machine    : client1.austin.ibm.com
      Handle Name: STAF/Client
      Handle     : 25
    }
    Pending Notifications: [
      {
        Machine           : client2.austin.ibm.com
        Notify By         : Name
        Notifiee          : SpanishFVTTest
        Attempts Remaining: 0
        Timeout           : 60000
        Priority          : 4
        Priority Delta    : 1
      }
      {
        Machine           : client3.austin.ibm.com
        Notify By         : Name
        Notifiee          : SpanishSVTTest
        Attempts Remaining: 0
  ```

```
          Timeout            : 60000
          Priority           : 4
          Priority Delta     : 1
        }
      ]
    }
```

RESET

RESET clears all information about the registered and/or generated events.

Syntax

```
RESET  <REG | GEN> FORCE
```

`REG` indicates that all information about the currently registered events is to be cleared.

`GEN` indicates that all information about the currently generated events is to be cleared.

`FORCE` is a confirmation that you want the specified information to be cleared.

Security

This command requires trust level 4.

Results

On a successful reset, the result buffer will be empty.

Examples

- **Goal:** Clear all information about the currently registered events.

  ```
  RESET REG FORCE
  ```

- **Goal:** Clear all information about the currently generated events.

  ```
  RESET GEN FORCE
  ```

VERSION

VERSION displays the Event Service version.

Syntax

```
VERSION
```

Security

This request requires at least trust level 1.

Results

The result is the version number of the Event service.

Examples

- **Goal:** Display the version of the Event service on machine server1.company.com:

  ```
  STAF server1.company.com EVENT VERSION
  ```

  Output:

  ```
  3.1.4
  ```

HELP

HELP displays the request options and how to use them.

Syntax

```
HELP
```

Security

This request requires at least trust level 1.

Results

The result buffer contains the Help messages for the request options for the Event service.

Examples

- **Goal:** Display the syntax for the EVENT service requests:

  ```
  STAF local EVENT HELP
  ```

  Output:

  ```
  EVENT Service Help

  REGISTER           TYPE <Type> [SUBTYPE <Subtype>]...
                     [ACKNOWLEDGETIMEOUT <Timeout>] [MAXATTEMPTS <Number>]
                     PRIORITY <Number>] [PRIORITYDELTA <Number>]
                     [BYNAME | BYHANDLE]

  UNREGISTER         TYPE <Type> [SUBTYPE <Subtype>]...
  ```

```
                       [FORCE [MACHINE <Machine>] [HANDLE <Handle> | NAME <Name>]]

ACKNOWLEDGE            EVENTID <Event ID>
                       [FORCE [MACHINE <Machine>] [HANDLE <Handle> | NAME <Name>]]

GENERATE               TYPE <Type> SUBTYPE <Subtype> [PROPERTY <Name>[=<Value>]]...

LIST                   TYPES [LONG]
LIST                   SUBTYPES TYPE <Type>
LIST                   REGISTRATIONS [TYPE <Type> [SUBTYPE <Subtype>]] [LONG]
LIST                   EVENTIDS [LONG]
LIST                   SETTINGS

QUERY                  EVENTID <Event ID> [LONG]

RESET                  <REG | GEN> FORCE

VERSION

HELP
```

Error Code Reference

In addition to the common STAF return codes (see Appendix A, "API Return Codes" in the STAF User's Guide for additional information), the following Event Service return codes are defined:

Error Code	Meaning	Comment
4001	No Acknowledgement Pending	Event Service has already received an acknowledgement for this event from the acknowledgeing process
4002	No such event ID	An acknowledgement was received but no event with the submitted Event ID has been generated.
4003	Not Registered for Type	A process tried to unregister for an event Type that it was not currently registered for.
4004	Not Registered for Subtype	A process tried to unregister for an event Subype that it was not currently registered for.
4005	No clients registered	No clients were registered for the event.

EventManager Service

Using the EventManager service can simplify using the Event service. The EventManager service allows you to register with the Event service in order to execute a STAF service request when an event type/subtype is generated. When an event is generated, the Event Service sends a STAF/Service/Event message to the EventManager service machine that registered for the event and the EventManager submits the STAF service request that was registered for the event type/subtype. This way, you don't have to write a program to listen for STAF/Service/Event messages on your queue because the EventManager service does this for you. The EventManager also provides a User Interface to simplify interaction with the EventManager service.

The EventManager service is a STAF Java service available via http://staf.sourceforge.net/getservices.php#EventManager. For more information on the EventManager service, see the EventManager Service User's Guide. Also, for an example of how you can use the EventManager service to simplify interaction with the Event service, see Example 2: Using the EventManager Service in this document.

Examples

Here are a couple of examples that demonstrate how you can use the Event service.

Example 1: Using the Event Service via Two Java Applications

This example shows how one process (in this case, a Java application) creates new builds for a product named MyApp. Whenever it generates a new build, it submits a GENERATE request to the Event service. The GENERATE request must specify the event type and subtype of the event it is generating via the TYPE and SUBTYPE options. Optionally, it can specify one or more properties if you want to provide additional information in the event message that gets generated. Whenever an event is generated, the Event service creates an event ID that is associated with the event. This example has a second process (in this case, another Java application) that wants to be notified when these build events occurs. So, it submits a REGISTER request to the Event service, specifying the event type for which it is registering, and optionally, it can specify an event subtype if it only wants to be notified when a particular event subtype is generated. You can also specify whether to register to be notified by your handle's name (via the BYNAME option, which is the default) or by your handle's number (via the BYHANDLE option). When an event with this type/subtype is generated, a message about this event will be automatically sent to the queue of the handle(s) that are registered to be notified about this event. The process(es) whose STAF handle is sent the event message can get the message by submitting a GET request to the QUEUE service. Generally, you'd have your process run a separate thread that has a loop to continually receive event messages by submitting a GET WAIT request to the QUEUE service on the local machine so that when a message appears on your queue the QUEUE GET WAIT request completes and you can process that message and then continue on in the loop to wait for more messages on your queue.

The first Java application is named MyAppBuild.java and generates two events when it creates a build for a product named MyApp:

1. An event with type *build* and subtype *win32* with a couple of properties (buildFile, buildType)
2. An event with type *build* and subtype *linux* with a couple of properties (buildFile, buildType)

The second Java application is named MyAppTest.java and it registers to be notified when any event with type "build" is generated. It runs a thread that listens for STAF/Service/Event type messages and for each message with event type *build*, prints a message about the STAF/Service/Event type message it received on its queue. Note that in a real example, it would do something more useful when it receives a event message such as installing this build of MyApp on a test machine and running tests on it.

To run this example, compile both files and then from one command prompt, run MyAppTest, and from another command prompt, run MyAppBuild.java. Note that you can kill MyAppTest when the test completes to end it.

For example (assuming the directory containing MyAppBuild.class and MyAppTest.class is in the CLASSPATH), here's the output you could get running these two Java applications. Note that if you run these Java applications on a machine other than the machine where you have the Event service registered, you'll need to modify the two .java files to set fEventServiceName to the host name or IP address of the machine the Event service is registered.

```
C:\>java MyAppTest

Registered to be notified when event type 'build' with any subtype is generated

Starting thread to listen for STAF/Service/Event messages...

Received STAF/Service/Event message:
Event Service : Event
Event ID : 24
Event Type : build
Event Subtype : win32
Generating Machine: local://local
Generating Process: MyApp/Build
Generating Handle : 64
Event Timestamp : 20090430-15:01:36
Properties : {buildID=MyApp123, buildFile=C:/Builds/MyApp-win32.zip}

Received STAF/Service/Event message:
Event Service : Event
Event ID : 25
Event Type : build
Event Subtype : linux
Generating Machine: local://local
Generating Process: MyApp/Build
Generating Handle : 64
Event Timestamp : 20090430-15:01:41
Properties : {buildID=MyApp456, buildFile=C:/Builds/MyApp-linux.zip}

C:\>java MyAppBuild

Building a Windows 32-bit build for MyApp...
Generated event ID 24 for the win32 build for MyApp

Building a Linux build for MyApp...
Generated event ID 25 for the linux build for MyApp
```

Here's the source code for MyAppBuild.java:

```
import com.ibm.staf.*;
import java.util.*;
import java.io.*;

public class MyAppBuild
{
    // Specify the machine where the Event service is registered
    private String fEventServiceMachine = "local";

    private STAFHandle fHandle = null;
    private String eventType = "build";

    public static void main(String[] args)
    {
        MyAppBuild build = new MyAppBuild();

        try
        {
            // Pretend that it is building a Windows 32-bit build for MyApp

            System.out.println(
                "\nBuilding a Windows 32-bit build for MyApp...");
            Thread.sleep(5000);  // Sleep for 5 seconds

            // Generate an event with type build and subtype win32 to indicate
            // that a Windows 32-bit build for MyApp has been created.
            // Also, specify some properties like the name of the build file
            // created and its build id.

            build.generateEvent(
                "win32", "C:/Builds/MyApp-win32.zip", "MyApp123");

            // Pretend that it is building a Linux build for MyApp

            System.out.println(
                "\nBuilding a Linux build for MyApp...");
            Thread.sleep(5000);  // Sleep for 5 seconds

            // Generate an event with type build and subtype linux to indicate
            // that a Linux build for MyApp has been created.
            // Also, specify some properties like the name of the build file
            // created and its build id.

            build.generateEvent(
                "linux", "C:/Builds/MyApp-linux.zip", "MyApp456");
        }
        catch (Exception e)
        {
            System.out.println("Exception: " + e.toString());
            System.exit(1);
        }
    }

    public MyAppBuild()
    {
        // Register with STAF to create a STAF handle

        try
```

```
            {
                fHandle = new STAFHandle("MyApp/Build");
            }
            catch (STAFException e)
            {
                System.out.println("Exception: " + e.toString());
                System.exit(1);
            }
        }

        private void generateEvent(String eventSubtype, String buildFile,
                                   String buildID)
        {
            // Generate an event with type build and the specified subtype to
            // indicate that a build for MyApp has been created.  Also,
            // the buildFile and buildID properties (these can be whatever
            // properties you want to provide).

            String request = "GENERATE TYPE " + STAFUtil.wrapData(eventType) +
                " SUBTYPE " + STAFUtil.wrapData(eventSubtype) +
                    " PROPERTY " + STAFUtil.wrapData("buildFile=" + buildFile) +
                    " PROPERTY " + STAFUtil.wrapData("buildID=" + buildID);

            STAFResult res = fHandle.submit2(
                fEventServiceMachine, "Event", request);

            if (res.rc == STAFResult.Ok)
            {
                System.out.println(
                    "Generated event ID " + res.result + " for the " +
                    eventSubtype + " build for MyApp");
            }
            else
            {
                System.out.println(
                    "STAF " + fEventServiceMachine + " Event " + request +
                    "\nFailed with RC: " + res.rc + ", Result: " + res.result);
            }
        }
    }
```

Here's the source code for MyAppTest.java:

```
    import com.ibm.staf.*;
    import java.util.*;
    import java.io.*;

    public class MyAppTest
    {
        // Specify the machine where the Event service is registered
        private String fEventServiceMachine = "local";
        private STAFHandle fHandle = null;
        private MonitorThread fMonitorThread = null;
        private String eventType = "build";

        public static void main(String[] args)
        {
            MyAppTest test = new MyAppTest();
        }

        public MyAppTest()
        {
            // Register with STAF to create a STAF handle

            try
            {
                fHandle = new STAFHandle("MyApp/Test");
            }
            catch (STAFException e)
            {
                System.out.println("Exception: " + e.toString());
                System.exit(1);
            }

            try
            {
                // Register with the Event service to be notified when event type
                // "build" with any subtype is generated.
                // Note this only has to be done once (before any events with this
                // type are generated).

                String request = "REGISTER TYPE " + STAFUtil.wrapData(eventType);

                STAFResult res = fHandle.submit2(
                    fEventServiceMachine, "Event", request);

                if (res.rc == STAFResult.Ok)
                {
                    System.out.println(
                        "\nRegistered to be notified when event type '" +
                        eventType + "' with any subtype is generated");
                }
                else
                {
                    System.out.println(
                        "STAF " + fEventServiceMachine + " Event " + request +
                        "\nFailed with RC: " + res.rc + ", Result: " + res.result);
                }

                // Start a thread to listen for STAF/Service/Event messages on
                // this application's STAF handle's queue

                System.out.println(
                    "\nStarting thread to listen for STAF/Service/Event " +
                    "messages...");

                fMonitorThread = new MonitorThread(fHandle);
                fMonitorThread.start();
            }
            catch (Exception e)
            {
                System.out.println("Exception: " + e.toString());
                System.exit(1);
            }
        }

        /**
         * This helper class thread loops continuously, waiting to get
         * messages on its queue with queue type STAF/Service/Event
         */
        class MonitorThread extends Thread
        {
            private STAFHandle fHandle;

            MonitorThread(STAFHandle handle)
            {
                fHandle = handle;
            }

            public void run()
            {
                STAFResult getResult = new STAFResult();
                Map queueMessageMap = null;

                for (;;)
                {
                    // Need a try/catch block in case an error occurs getting
                    // a message off the queue so we can continue processing
                    // additional messages

                    try
                    {
                        getResult = fHandle.submit2(
                            "local", "QUEUE", "GET WAIT");

                        if (getResult.rc != 0)
                        {
                            System.out.println(
                                "ERROR: STAF local QUEUE GET WAIT failed." +
                                "\nRC: " + getResult.rc +
                                ", Result: " + getResult.result);
                            continue;
                        }

                        queueMessageMap = (Map)getResult.resultObj;
                        String queueType = (String)queueMessageMap.get("type");

                        if (queueType == null)
                        {
                            continue;
                        }
                        else if (queueType.equalsIgnoreCase("STAF/Service/Event"))
                        {
                            if (!(queueMessageMap.get("message") instanceof Map))
                            {
                                System.out.println(
                                    "Unsupported message format.  " +
                                    "Ignoring this message.
    \n");
                                continue;
                            }

                            handleEventServiceMessage(
                                (Map)queueMessageMap.get("message"));
                        }
                        else
                        {
                            // Ignore other messages
                        }
                    }
                    catch (Exception e)
                    {
                        System.out.println(
                            "Exception handling queued message. Exception:\n" +
                            e.toString() +
                            "\nQueued Message:\n" +
    queueMessageMap);
                        e.printStackTrace();
                    }
                }
            }

            private void handleEventServiceMessage(Map messageMap)
            {
                // A STAF/Service/Event message is a map containing keys:
                // eventServiceName, eventID, machine, handleName, handle,
                // timestamp, type, subtype, and propertyMap

                String eventService = (String)messageMap.get("eventServiceName");
                String eventID = (String)messageMap.get("eventID");
                String generatingMachine = (String)messageMap.get("machine");
                String generatingProcess = (String)messageMap.get("handleName");
                String generatingHandle = (String)messageMap.get("handle");
                String eventTimestamp = (String)messageMap.get("timestamp");
                String eventType = (String)messageMap.get("type");
                String eventSubtype = (String)messageMap.get("subtype");
                Map eventProperties = (Map)messageMap.get("propertyMap");
```

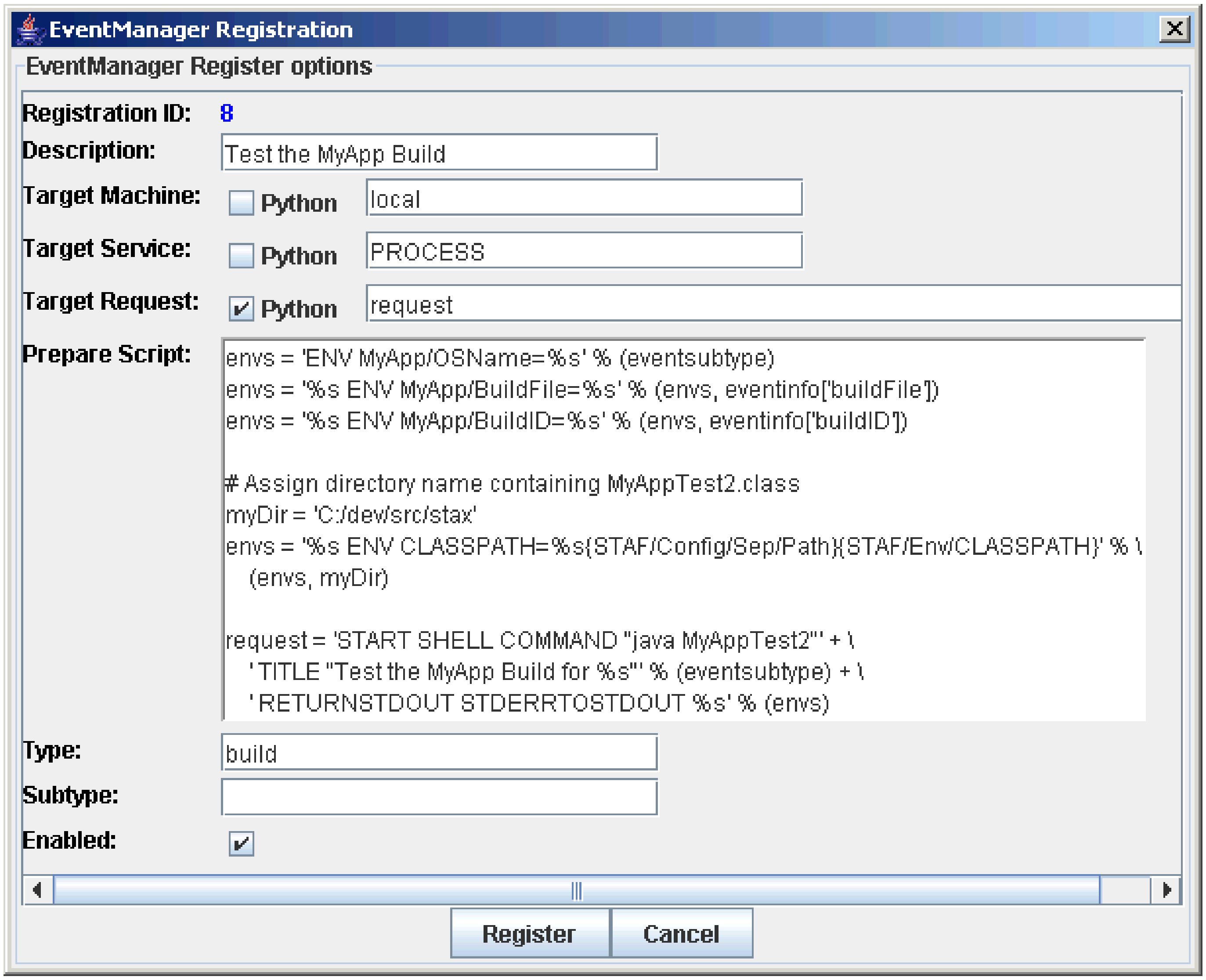
```
                System.out.println(
                    "\nReceived STAF/Service/Event message: " +
                    "\n  Event Service     : " + eventService +
                    "\n  Event ID          : " + eventID +
                    "\n  Event Type        : " + eventType +
                    "\n  Event Subtype     : " + eventSubtype +
                    "\n  Generating Machine: " + generatingMachine +
                    "\n  Generating Process: " + generatingProcess +
                    "\n  Generating Handle : " + generatingHandle +
                    "\n  Event Timestamp   : " + eventTimestamp +
                    "\n  Properties        : " + eventProperties);

                if (eventType.equals("build"))
                {
                    String buildFile = (String)eventProperties.get("buildFile");
                    String buildID = (String)eventProperties.get("buildID");

                    // Could do something useful here like get the build for MyApp,
                    // install the MyApp build on a test machine and run tests
                    // on it
                }
            }
        }
    }
```

Example 2: Using the EventManager Service

Instead of writing a program that registers for an Event type/subtype and continuously gets messages off it's queue, you could use the EventManager service to register with the Event service and listen for events. The EventManager service allows you to register with the Event service in order to execute a STAF service request when an event type/subtype is generated. When an event is generated, the Event Service sends a STAF/Service/Event message to the EventManager service machine and the EventManager submits the STAF service request that was registered for the event type/subtype. The EventManager provides a User Interface to simplify interaction with the EventManager service. For more information on the EventManager service, see the EventManager Service User's Guide.

Here's an example that uses the EventManager service to register for Event type *build* and to submit a STAF service request that runs a Java application named MyAppTest2 via a PROCESS START request when it gets notified that a build for MyApp is available. This example merely logs this information, but it could be changed to do something more useful like get the build for MyApp and install it on a test machine and run tests.

Here's how you could register with the EventManager service via its User Interface for event type *build* to run the Java application MyAppTest2. It uses the Python Prepare Script to generate the PROCESS START request using Python variables provided. For example, the subtype is provided via Python variable eventsubtype (note that in this example, the event subtype contains the operating system name for the build). In addition, it shows how to obtain any property values that were provided via the `eventInfo` Python dictionary (aka map). For example, it obtains the `buildFile` property via Python variable `eventInfo['buildFile']` and obtains the `buildID` property via Python variable `eventInfo['buildID']`. It provides these values as environment variables using the ENV option for a PROCESS START request (though you could have implemented this using a different method) so that the MyAppTest2 Java application can access these values as environment variables.



Registration ID: 8
Description: Test the MyApp Build
Target Machine: ☐ Python local
Target Service: ☐ Python PROCESS
Target Request: ✓ Python request
Prepare Script:

```
envs = 'ENV MyApp/OSName=%s' % (eventsubtype)
envs = '%s ENV MyApp/BuildFile=%s' % (envs, eventinfo['buildFile'])
envs = '%s ENV MyApp/BuildID=%s' % (envs, eventinfo['buildID'])

# Assign directory name containing MyAppTest2.class
myDir = 'C:/dev/src/stax'
envs = '%s ENV CLASSPATH=%s{STAF/Config/Sep/Path}{STAF/Env/CLASSPATH}' % \
   (envs, myDir)

request = 'START SHELL COMMAND "java MyAppTest2"' + \
   'TITLE "Test the MyApp Build for %s"' % (eventsubtype) + \
   'RETURNSTDOUT STDERRTOSTDOUT %s' % (envs)
```

Type: build
Subtype:
Enabled: ✓

Note that you could have any STAF service request run instead of submitting a PROCESS START request to run a Java application. For example, if you preferred, you could change the EventManager registration to submit a STAX EXECUTE request to run a STAX job.

Here's the source code for the Java application, MyAppTest2.java, that is run whenever an event with type *build* (and any subtype) is generated.

```
public class MyAppTest2
{
    public static void main(String[] args)
    {
        MyAppTest2 test = new MyAppTest2();
    }

    public MyAppTest2()
    {
        // The following environment variables should be set:
        // - MyApp/OSName:  The operating system for the MyApp build
        // - MyApp/BuildFile:  The name of the file containing the build
        //   for MyApp for this operating system
        // - MyApp/BuildID:  The build ID
        String osName = System.getenv("MyApp/OSName");
        String buildFile = System.getenv("MyApp/BuildFile");
        String buildID = System.getenv("MyApp/BuildID");

        System.out.println(
            "\nReceived MyApp build event: " +
            "\n  MyApp/OSName   : " + osName +
            "\n  MyApp/BuildFile: " + buildFile +
            "\n  MyApp/BuildID  : " + buildID);

        // Could do something use here like get the build for MyApp,
        // install the MyApp build on a test machine and run tests
        // on it
    }
}
```

You could generate build events for MyApp by running Java application MyAppBuild from Example 1: Using the Event Service via Two Java Applications. Here are the EventManager service log entries you could see when two events are generated by MyAppBuild, one for a win32 build of MyApp and another for a linux build of MyApp.

```
C:\>STAF -verbose local LOG QUERY MACHINE {STAF/Config/MachineNickname} LOGNAME
EventManager STARTSWITH "[ID=8]"
Response
--------
[
  {
    Date-Time: 20090504-15:19:07
    Level    : Info
    Message  : [ID=8] [local://local, STAF/EventManager/UI] Registered a STAF co
mmand.
Register request: REGISTER MACHINE :5:local SERVICE :7:PROCESS PYTHONREQUEST :7:
request PREPARE :515:envs = 'ENV MyApp/OSName=%s' % (eventsubtype)
envs = '%s ENV MyApp/BuildFile=%s' % (envs, eventinfo['buildFile'])
envs = '%s ENV MyApp/BuildID=%s' % (envs, eventinfo['buildID'])

# Assign directory name containing MyAppTest2.class
myDir = 'C:/dev/src/stax'
envs = '%s ENV CLASSPATH=%s{STAF/Config/Sep/Path}{STAF/Env/CLASSPATH}' % \
    (envs, myDir)

request = 'START SHELL COMMAND "java MyAppTest2"' + \
    ' TITLE "Test the MyApp Build for %s"' % (eventsubtype) + \
    ' RETURNSTDOUT STDERRTOSTDOUT %s' % (envs) TYPE :5:build DESCRIPTION :20:Tes
t the MyApp Build
  }
  {
    Date-Time: 20090504-15:32:25
    Level    : Info
    Message  : [ID=8] [mach1.austin.ibm.com:235] Submitted a STAF command.
Event information: type=build subtype=win32 prepare=envs = 'ENV MyApp/OSName=%s'
```

```
 % (eventsubtype)
envs = '%s ENV MyApp/BuildFile=%s' % (envs, eventinfo['buildFile'])
envs = '%s ENV MyApp/BuildID=%s' % (envs, eventinfo['buildID'])

# Assign directory name containing MyAppTest2.class
myDir = 'C:/dev/src/stax'
envs = '%s ENV CLASSPATH=%s{STAF/Config/Sep/Path}{STAF/Env/CLASSPATH}' % \
    (envs, myDir)

request = 'START SHELL COMMAND "java MyAppTest2"' + \
    ' TITLE "Test the MyApp Build for %s"' % (eventsubtype) + \
    ' RETURNSTDOUT STDERRTOSTDOUT %s' % (envs) eventservice=Event eventid=3 gene
ratingmachine=local://local generatingprocess=MyApp/Build generatinghandle=33 ev
enttimestamp=20090504-15:32:24 properties={buildID=MyApp123, buildFile=C:/Builds
/MyApp-win32.zip}
STAF command: STAF local PROCESS START NOTIFY ONEND HANDLE 23 SHELL COMMAND "jav
a MyAppTest2" TITLE "Test the MyApp Build for win32" RETURNSTDOUT STDERRTOSTDOUT
 ENV MyApp/OSName=win32 ENV MyApp/BuildFile=C:/Builds/MyApp-win32.zip ENV MyApp/
BuildID=MyApp123 ENV CLASSPATH=C:/dev/src/stax{STAF/Config/Sep/Path}{STAF/Env/CL
ASSPATH}
  }
  {
    Date-Time: 20090504-15:32:25
    Level    : Pass
    Message  : [ID=8] [mach1.austin.ibm.com:235] Completed a STAF command. RC=0,
 Result=34
  }
  {
    Date-Time: 20090504-15:32:26
    Level    : Pass
    Message  : [ID=8] [mach1.austin.ibm.com:235] Process completed.
{
  Return Code: 0
  Key        :
  Files      : [
    {
      Return Code: 0
      Data       :
Received MyApp build event:
  MyApp/OSName   : win32
  MyApp/BuildFile: C:/Builds/MyApp-win32.zip
  MyApp/BuildID  : MyApp123

    }
  ]
}
  }
  {
    Date-Time: 20090504-15:32:29
    Level    : Info
    Message  : [ID=8] [mach1.austin.ibm.com:253] Submitted a STAF command.
Event information: type=build subtype=linux prepare=envs = 'ENV MyApp/OSName=%s'
```

```
     % (eventsubtype)
    envs = '%s ENV MyApp/BuildFile=%s' % (envs, eventinfo['buildFile'])
    envs = '%s ENV MyApp/BuildID=%s' % (envs, eventinfo['buildID'])

    # Assign directory name containing MyAppTest2.class
    myDir = 'C:/dev/src/stax'
    envs = '%s ENV CLASSPATH=%s{STAF/Config/Sep/Path}{STAF/Env/CLASSPATH}' % \
        (envs, myDir)

    request = 'START SHELL COMMAND "java MyAppTest2"' + \
        ' TITLE "Test the MyApp Build for %s"' % (eventsubtype) + \
        ' RETURNSTDOUT STDERRTOSTDOUT %s' % (envs) eventservice=Event eventid=4 gene
    ratingmachine=local://local generatingprocess=MyApp/Build generatinghandle=33 ev
    enttimestamp=20090504-15:32:29 properties={buildID=MyApp456, buildFile=C:/Builds
    /MyApp-linux.zip}
    STAF command: STAF local PROCESS START NOTIFY ONEND HANDLE 23 SHELL COMMAND "jav
    a MyAppTest2" TITLE "Test the MyApp Build for linux" RETURNSTDOUT STDERRTOSTDOUT
     ENV MyApp/OSName=linux ENV MyApp/BuildFile=C:/Builds/MyApp-linux.zip ENV MyApp/
    BuildID=MyApp456 ENV CLASSPATH=C:/dev/src/stax{STAF/Config/Sep/Path}{STAF/Env/CL
    ASSPATH}
      }
      {
        Date-Time: 20090504-15:32:30
        Level    : Pass
        Message  : [ID=8] [mach1.austin.ibm.com:253] Completed a STAF command. RC=0,
     Result=35
      }
      {
        Date-Time: 20090504-15:32:30
        Level    : Pass
        Message  : [ID=8] [mach1.austin.ibm.com:253] Process completed.
    {
      Return Code: 0
      Key        :
      Files      : [
        {
          Return Code: 0
          Data       :
    Received MyApp build event:
      MyApp/OSName   : linux
      MyApp/BuildFile: C:/Builds/MyApp-linux.zip
      MyApp/BuildID  : MyApp456

        }
      ]
    }
      }
    ]
```