



iControl[®] REST for
BIG-IP[®] Advanced Firewall
Manager[™]



Legal Notices

Copyright © 2015, F5 Networks, Inc. All rights reserved.

F5 Networks, Inc. (F5) believes the information it furnishes to be accurate and reliable. However, F5 assumes no responsibility for the use of this information, nor any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent, copyright, or other intellectual property right of F5 except as specifically described by applicable user licenses. F5 reserves the right to change specifications at any time without notice.

Trademarks

Advanced Firewall Manager, BIG-IP, F5, F5 [DESIGN], F5 Networks, iControl, iRules, and TMOS are trademarks or service marks of F5 Networks, Inc., in the U.S. and other countries, and may not be used without F5's express written consent.

All other product and company names herein may be trademarks of their respective owners.

This document introduces basic AFM Access Control Rules principles and configuration, then explains how to manage the ACLS by using the iControl REST API. Markings are best seen when printed in color.

I-Introduction

II-Basics of iControl REST API & tmsh

III-AFM iControl REST API

IV-Conclusion

I INTRODUCTION

BIG-IP® system is a comprehensive platform that can be expanded with feature-based modules chosen by user needs. BIG-IP Advanced Firewall Manager™ (AFM™) is a full proxy network firewall that is offered in the BIG-IP product family.

USER INTERFACES

BIG-IP system has three main user-facing interfaces that allow the management and configuration of AFM: tmsh (Traffic Management Shell), which is a command line interface, TMUI (Traffic Management User Interface), which is a web-based interface, and iControl® REST API, which is an HTTP REST-based programmatic interface. In terms of capabilities, tmsh offers advanced commands and power-user functions that may not be available in tmui. In the background tmui relies on iControl REST API.

tmsh is the core player in iControl REST management of the product. It serves as a gateway to the system configuration. It understands iControl REST requests, converts them into meaningful tmsh commands, and generates a JSON response.

TMSH

For one to understand iControl REST, it is vital to understand basic tmsh. tmsh utilizes a hierarchical command structure that is composed of the following elements:



Action: create/modify/delete/show/list

Module: security (AFM)

SubModule: firewall (ACL-Access Control List)

Component: policy

Objects: {...}

? **TIP:** Any command that can be run in tmsh is also accessible using the iControl REST API. When in doubt, tmsh can be used to check whatever commands are available under a module or whether the commands you're running are valid.

The above tmsh command structure can be used as a guideline to derive iControl REST versions of the requests very easily. tmsh commands, alone, may not be always very definitive, but when used along with iControl REST response content (link, selflink, etc.), it gives enough information to the user. This will be explained in detail in the next few sections.

Below are six basic tmsh commands to manage AFM ACL global policies, which we will use in our examples later in this document.

	Action	Module	SubModule	Component	Objects
1	<code>create</code>	<code>security</code>	firewall	policy	<code><policy_name></code>
2	<code>modify</code>	<code>security</code>	firewall	policy	<code><policy_name></code> <div style="border: 1px solid black; padding: 5px; margin-left: 20px;"> rules add { ... } modify { ... } delete { ... } </div>
3	<code>modify</code>	<code>security</code>	firewall	global-rules <div style="margin-left: 20px;">↳ <code>enforced/staged-policy</code></div>	<code><policy_name></code>
4	<code>show</code>	<code>security</code>	firewall	policy	<code><policy_name></code>
5	<code>delete</code>	<code>security</code>	firewall	policy	<code><policy_name></code>
6	<code>list</code>	<code>security</code>	firewall	global-rules <div style="margin-left: 20px;">↳ <code>enforced/staged-policy</code></div>	<code><policy_name></code>

Figure 1 Six basic tmsh commands

Command descriptions:

1. Create a policy `<policy_name>`.
2. Add/delete rules or modify existing rules.
3. Apply (aka “attach”) `<policy_name>` to global context.
4. Show the contents of the policy `<policy_name>`.
5. Delete the policy `<policy_name>`.
6. List currently enforced/staged policy for global context.

AFM ACL BASICS

Context: It is very common to hear about AFM firewall contexts, but what exactly is a context? *Context* in AFM is the category of the object to which firewall rules apply. The context can be Global, Route Domain, Self IP or Virtual Server. Attaching firewall rules to a Global context implies that those rules should be applied to all traffic traversing the AFM firewall. The next sections will use the Global context as the context in the examples.

Firewall Rules can also be attached to Route Domain, Self IPs and Virtual Server Contexts. In those cases, the rules will be limited in scope to only a specific set of VLANs/interfaces, or only for specific destination IP subnets and ports.

Policy: A basic collection of firewall rules. Policies can be thought as books and rules as its pages. Policies can be either standalone or applied to contexts. When they are standalone, they have no effect. The user creates policies, adds rules to them, and then applies to a context. They can be applied as *enforced* or *staged*. Enforced policies take action when there is a match. Staged policies don’t take action but they increase statistical counters so as to

give the customer an idea about their possible effects. We will concentrate on enforced policies as staged policies are optional, and enforced policies are mostly the main method used.

Rules: Firewall rules are minimally composed of actions, a rule name, and a relative place directive that specifies order among other rules.

Partition: A BIG-IP system object to define separate configuration locations. This offers granularity and security. For example, a firewall manager can access all partitions. If we wanted to limit a specific user from accessing some sensitive configuration, such configuration could be created in a new partition. Only users allowed access to that partition could see/change the configuration. All the policies can be simply put in Common partition (it's the default partition where a policy is created, so no action is needed other than knowing the concept itself).

II Basics of iControl REST API & tmsh

iControl REST Basics

In the previous section, we introduced six basic tmsh commands. In this section, we will see how they're translated into iControl REST requests. Then we will examine the returned output. This section can be read as a tutorial. The reader can execute these, step by step to understand the simplicity of the system.

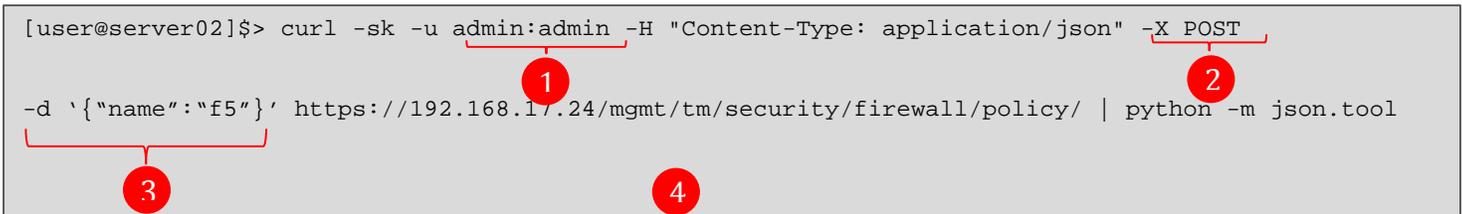
iControl REST uses camelCase naming convention. For example, in command #3, the tmsh property "enforced-policy" translates into "enforcedPolicy" in iControl REST. The tmsh component "global-rules" becomes "globalRules" in iControl REST.

Exception: For user-defined properties (for example, rule name, policy name, etc), this rule doesn't apply.

For the simplicity of this document, we will use simple curl queries to run the six tmsh commands, shown in Figure 1, using the iControl REST API.

A basic REST query format looks like this:

```
[user@server02]$> curl -sk -u admin:admin -H "Content-Type: application/json" -X POST  
-d '{"name": "f5"}' https://192.168.17.24/mgmt/tm/security/firewall/policy/ | python -m json.tool
```



1. Credentials

2. HTTP Method

3. Request Body

4. URI

tmsh to iControl REST TRANSLATION

Managing AFM using iControl REST is extremely trivial. Once we decide on which tmsh command to run, the following steps are observed.

1. Credentials: BIG-IP system credentials. Default value is “*admin:admin*” for all systems.

2. HTTP Method and URI formatting: HTTP methods are determined by looking at the tmsh command action. Also notice the tmsh structural command elements in URI below.

```
TMSH create = HTTP POST to the URI ...tm/<module>/<submodule>/<component>
TMSH modify = HTTP PATCH to the URI ...tm/<module>/<submodule>/<component>/<objid>
TMSH delete = HTTP DELETE to the URI ...tm/<module>/<submodule>/<component>/<objid>
TMSH show = HTTP GET to the URI ...tm/<module>/<submodule>/<component>/<subCollection>
TMSH list = HTTP GET to the URI ...tm/<module>/<submodule>/<component> (to get all)
           to the URI ...tm/<module>/<submodule>/<component>/<objid> (to get specific)
```

3. Request Body: Needed when POST or PATCH is used. Contains object related information. It’s specified after –d parameter, please pay special attention to punctuation:

```
-d '{"property_name":"property_value", "property_name_2":" property_value_2", ... }'
```

4. URI: For BIG-IP systems, the URI format always starts with “*https://xxx.xxx.xxx.xxx/mgmt/tm/...*”, where x’s stand for the management IP address of the BIG-IP system. The rest of the URI changes according to the tmsh command that’s about to be executed. See #2 above for the general guidelines.



TRICK: Judging a tmsh command can sometimes be confusing if properties are included in a command. In those cases, the tmsh interface is very handy. To determine the previously described structural command elements (module, submodule, component, object, etc) in a tmsh command, throw away the action part from the command, and enter the rest of the keywords one by one in tmsh. Once tmsh reaches the component element, it won’t accept keywords anymore.

Example: *modify security firewall global-rules enforced-policy iCtrPolicyExample*

```
root@(localhost)(cfg-sync Standalone)(Active)(/Common)(tmos)# security
root@(localhost)(cfg-sync Standalone)(Active)(/Common)(tmos.security)# firewall
root@(localhost)(cfg-sync Standalone)(Active)(/Common)(tmos.security.firewall)# global-rules
root@(localhost)(cfg-sync Standalone)(Active)(/Common)(tmos.security.firewall.global-rules)# enforced-policy
Syntax Error: unexpected argument "enforced-policy"
```

Our component is “global-rules”. In this case “enforced-policy” is our property. Remember to use camelCasing. We are modifying “global-rules” to change its policy, so intuitively REST method is PATCH (also because tmsh action is modify). Hence the final query will be as follows:

```
curl -sk -u admin:admin -H "Content-Type: application/json" -X PATCH -d
'{"enforcedPolicy":"iCtrPolicy"}' https://192.168.17.24/mgmt/tm/security/firewall/globalRules/ |
python -m json.tool
```

iControl REST TRANSACTIONS

iControl REST API supports DBMS like transactions for executing multiple commands at once. This is especially handy in cases where one command depends on another. A transaction aggregates individual commands into one atomic operation. In case one of the commands fails, all other commands that were run are rolled back, ensuring atomicity. iControl REST transactions work on an all-or-none principle.

iControl REST transactions are identified by a transaction ID. It is automatically generated the first time a transaction is started. Once a transaction is started, the user adds all respective commands to this transaction ID, and then runs it.

```
curl -sk -u admin:admin -H "Content-Type: application/json" -X POST -d '{}'
https://192.168.17.24/mgmt/tm/transaction | python -m json.tool
```

Above command shows how a transaction is initiated. Pay special attention to markings. A POST request with an empty body sent to BIG-IP management IP address. Also notice “./tm/transaction” in URI. In response to the above request, the following response is returned:

```
{
  "asyncExecution": false,
  "executionTime": 0,
  "executionTimeout": 300,
  "failureReason": "",
  "kind": "tm:transactionstate",
  "selfLink": "https://localhost/mgmt/tm/transaction/1424381191992301?ver=1.0",
  "state": "STARTED",
  "timeoutSeconds": 30,
  "transId": 1424381191992301
}
```



“timeoutSeconds”: Amount of time passed until a transaction is terminated automatically.
“executionTimeOut”: Amount of time allowed for a transaction to complete. Once expired, it returns an error message.
“selfLink”: URI with transaction ID. It will be used for any proceeding operations concerning this transaction.
“transId”: Transaction ID.

Once a transaction is created, commands need to be added with POST method and HTTP header “X-F5-REST-Coordination-Id: xxx” needs to be specified, where x is the transaction ID. Examples can be found in the next chapter, command #8. The response received will be very similar to one above. If a transaction times out, 404 error code is returned.

iControl REST transactions are commonly used in scripts.

Below is a quick overview of URIs available while a transaction is active:

- HTTP GET to the URI `../tm/transaction/<transactionId>/commands` => lists all commands in a transaction.
- HTTP GET to the URI `../tm/transaction/<transactionId>/commands/2` => show the second command in a transaction.
- HTTP DELETE to the URI `../tm/transaction/<transactionId>/commands/5` => delete the fifth command in a transaction.

Once all commands are added, a transaction can be committed with *PATCH* method and HTTP header “X-F5-REST-Coordination-Id: xxx”, where x is transaction ID. Request body should contain “state”: “VALIDATING” to URI `../tm/transaction/<transactionId>`.



TIP: *icrd.conf* is an embedded BIG-IP file that contains various iControl configuration parameters. For instance, `timeoutSeconds` can be modified there. Default transaction timeout is set to 30 seconds.

III

AFM iControl REST API

In this chapter, all global-context REST commands are listed and run, and the query responses are explained.

1. Create policy: iCtrPolicy

TMSH: `create security firewall policy iCtrPolicy`

```
[user@server02 workspace]$ curl -sk -u admin:admin -H "Content-Type: application/json" -X POST -d '{"name":"iCtrPolicy"}' https://192.168.17.24/mgmt/tm/security/firewall/policy | python -m json.tool
```

```
{
  "fullPath": "iCtrPolicy",
  "generation": 868,
  "kind": "tm:security:firewall:policy:polycystate",
  "name": "iCtrPolicy",
  "rulesReference": {
    "isSubcollection": true,
    "link":
      "https://localhost/mgmt/tm/security/firewall/policy/~Common~iCtrPolicy/rules?ver=1.0.0"
  },
  "selfLink": "https://localhost/mgmt/tm/security/firewall/policy/iCtrPolicy?ver=1.0.0"
}
```



INFO: Above response shows details about the object that was just created along with some additional valuable information.

“rulesReference”: Shows that this policy refers to a *subCollection*, another container of objects. In fact this is what makes policy “a collection of rules”.

“link”: URI for the subcollection. It is used for modifying the objects of the subcollection. In this case rules are contained in subcollection. Note that how “rules” keyword is appended at the end in URI. This URI needs to be used for future operations (like modify/delete) regarding rule operations. Also note that in URI, partition name is wrapped around ~...~ and that by default policy was created in *Common* partition.

“selfLink” : URI for the object that was just created. In this case, URI for policy named *iCtrPolicy*. Notice that how policy name is appended at the end in URI. This URI needs to be used for future operations (like modify/delete) on this policy.

2. Create rules in iCtrPolicy

TMSH: modify security firewall policy iCtrPolicy rules add { myRule1 { action reject place-before first } }

```
[user@server02 workspace]$ curl -sk -u admin:admin -H "Content-Type: application/json" -X POST -d '{"name":"myRule1", "action":"reject", "place-before":"first"}' https://192.168.17.24/mgmt/tm/security/firewall/policy/~Common~iCtrPolicy/rules | python -m json.tool
{
  "action": "reject",
  "destination": {},
  "fullPath": "myRule1",
  "generation": 868,
  "ipProtocol": "any",
  "iruleSampleRate": 1,
  "kind": "tm:security:firewall:policy:rules:rulesstate",
  "log": "no",
  "name": "myRule1",
  "selfLink":
"https://localhost/mgmt/tm/security/firewall/policy/~Common~iCtrPolicy/rules/myRule1?ver=1.0.0",
  "source": {},
  "status": "enabled"
}
```

URI from step #1 was used to create a rule object in *rules* subcollection, which is referenced by policy *iCtrPolicy*.

? **TIP:** Firewall rules in a policy require at least a name, place directive and an action type specified. tmsh offers man pages under each component and a powerful tab completion tool to help identify what available/required properties there are under a command hierarchy.

3. Display rules in iCtrPolicy

TMSH: show security policy iCtrPolicy

```
[user@server02 workspace]$ curl -sk -u admin:admin -H "Content-Type: application/json" -X GET https://192.168.17.24/mgmt/tm/security/firewall/policy/~Common~iCtrPolicy/rules | python -m json.tool
{
  "items": [
    {
      "action": "accept",
      "destination": {},
      "fullPath": "myRule2",
      "generation": 896,
      "ipProtocol": "any",
      "iruleSampleRate": 1,
      "kind": "tm:security:firewall:policy:rules:rulesstate",
      "log": "no",
      "name": "myRule2",
      "selfLink":
"https://localhost/mgmt/tm/security/firewall/policy/~Common~iCtrPolicy/rules/myRule2?ver=1.0.0",
      "source": {},
      "status": "enabled"
    },
    {
      "action": "reject",
      "destination": {},
      "fullPath": "myRule1",
```

```

        "generation": 882,
        "ipProtocol": "any",
        "iruleSampleRate": 1,
        "kind": "tm:security:firewall:policy:rules:rulesstate",
        "log": "no",
        "name": "myRule1",
        "selfLink":
"https://localhost/mgmt/tm/security/firewall/policy/~Common~iCtrPolicy/rules/myRule1?ver=1.0.0",
        "source": {},
        "status": "enabled"
    }
],
"kind": "tm:security:firewall:policy:rules:rulescollectionstate",
"selfLink":
"https://localhost/mgmt/tm/security/firewall/policy/~Common~iCtrPolicy/rules?ver=1.0.0"
}

```

Judging the tmsh command, one might think that the URI should not have “/rules” at the end. However, running a GET request directly on the policy URI will return information specific to the policy object itself, not the rules in the subcollection where they’re stored. Remember, the REST response for creating a policy actually told us what URI to use in step #1.

4. Change rule action to ‘accept’ for myRule1

```
TMSH: security firewall policy iCtrPolicy rules modify { myRule1 { action accept } }
```

```
[user@server02 workspace]$ curl -sk -u admin:admin -H "Content-Type: application/json" -X
PATCH -d '{"action":"accept"}'
https://192.168.17.24/mgmt/tm/security/firewall/policy/~Common~iCtrPolicy/rules/myRule1 |
python -m json.tool
```

```

{
  "action": "accept",
  "destination": {},
  "fullPath": "myRule1",
  "generation": 868,
  "ipProtocol": "any",
  "iruleSampleRate": 1,
  "kind": "tm:security:firewall:policy:rules:rulesstate",
  "log": "no",
  "name": "myRule1",
  "selfLink":
"https://localhost/mgmt/tm/security/firewall/policy/~Common~iCtrPolicy/rules/myRule1?ver=1.0.0",
  "source": {},
  "status": "enabled"
}

```



TIP: While modifying objects, iControl REST API provides the flexibility of not defining all required properties for a rule. Note that above we did not specify either place directive or name even though they were required while creating rules, the first time. Whatever properties the user does not provide in the JSON body of the modified message, we will set to the default values if we know them (default could be empty string, some startup constants, etc.)

5. Delete rule myRule2 from iCtrPolicy

```
TMSH: modify security firewall policy iCtrPolicy rules delete { myRule1 }
```

```
[user@server02 workspace]$ curl -sk -u admin:admin -H "Content-Type: application/json" -X DELETE https://192.168.17.24/mgmt/tm/security/firewall/policy/~Common~iCtrPolicy/rules/myRule2 | python -m json.tool
```

No JSON object could be decoded



TIP: DELETE method does not return any JSON objects to parse. If the reply wasn't piped to JSON and `-v` flag was used in the request, we would have instead received HTTP 200 OK reply. In these examples, that output is suppressed.

6. Apply *iCtrPolicy* to Global context

```
TMSH: modify security firewall global-rules enforced-policy iCtrPolicy
```

```
[user@server02 workspace]$ curl -sk -u admin:admin -H "Content-Type: application/json" -X PATCH -d '{"enforcedPolicy": "iCtrPolicy"}' https://192.168.17.24/mgmt/tm/security/firewall/globalRules/ | python -m json.tool
```

```
{
  "enforcedPolicy": "/Common/iCtrPolicy",
  "enforcedPolicyReference":
"https://localhost/mgmt/tm/security/firewall/policy/~Common~iCtrPolicy?ver=1.0.0",
  "kind": "tm:security:firewall:global-rules:global-rulesstate",
  "selfLink": "https://localhost/mgmt/tm/security/firewall/global-rules?ver=1.0.0"
}
```

7. Display existing policy applied to Global context

```
TMSH: list security firewall global-rules enforced-policy
```

```
[user@server02 workspace]$ curl -sk -u admin:admin -H "Content-Type: application/json" -X GET https://192.168.17.24/mgmt/tm/security/firewall/globalRules | python -m json.tool
```

```
{
  "enforcedPolicy": "/Common/iCtrPolicy",
  "enforcedPolicyReference":
"https://localhost/mgmt/tm/security/firewall/policy/~Common~iCtrPolicy?ver=1.0.0",
  "kind": "tm:security:firewall:global-rules:global-rulesstate",
  "selfLink": "https://localhost/mgmt/tm/security/firewall/global-rules?ver=1.0.0"
}
```

8. Add two more rules to *iCtrPolicy* in a single transaction.

>START TRANSACTION

```
[user@server02 workspace]$ curl -sk -u admin:admin -H "Content-Type: application/json" -X POST -d '{} ' https://192.168.17.24/mgmt/tm/transaction | python -m json.tool
```

```
{
  "asyncExecution": false,
  "executionTime": 0,
  "executionTimeout": 300,
  "failureReason": "",
  "kind": "tm:transactionstate",
  "selfLink": "https://localhost/mgmt/tm/transaction/1424388547238937?ver=1.0.0",
  "state": "STARTED",
  "timeoutSeconds": 30,
  "transId": 1424388547238937
}
```

>ADD RULE

```
[user@server02 workspace]$ curl -sk -u admin:admin -H "Content-Type: application/json" -H "X-F5-REST-Coordination-Id:1424388547238937" -X POST -d '{"name":"iCtrRule2", "action":"reject", "place-after":"first"}' https://192.168.17.24/mgmt/tm/security/firewall/policy/~Common~iCtrPolicy/rules | python -m json.tool
```

```
{
  "body": {
    "action": "reject",
    "name": "iCtrRule2",
    "place-after": "first"
  },
  "commandId": 1,
  "evalOrder": 1,
  "kind": "tm:transaction:commandstate",
  "method": "POST",
  "selfLink": "https://localhost/mgmt/tm/transaction/1424388547238937/commands/1?ver=1.0.0",
  "uri": "https://localhost/mgmt/tm/security/firewall/policy/~Common~iCtrPolicy/rules"
}
```

>ADD RULE

```
[user@server02 workspace]$ curl -sk -u admin:admin -H "Content-Type: application/json" -H "X-F5-REST-Coordination-Id:1424388547238937" -X POST -d '{"name":"iCtrRule3", "action":"accept", "place-after":"iCtrRule2"}' https://192.168.17.24/mgmt/tm/security/firewall/policy/~Common~iCtrPolicy/rules | python -m json.tool
```

```
{
  "body": {
    "action": "accept",
    "name": "iCtrRule3",
    "place-after": "iCtrRule2"
  },
  "commandId": 2,
  "evalOrder": 2,
  "kind": "tm:transaction:commandstate",
  "method": "POST",
  "selfLink": "https://localhost/mgmt/tm/transaction/1424388547238937/commands/2?ver=1.0.0",
  "uri": "https://localhost/mgmt/tm/security/firewall/policy/~Common~iCtrPolicy/rules"
}
```

>COMMIT

```
[user@server02 workspace]$ curl -sk -u admin:admin -H "Content-Type: application/json" -X PATCH -d '{"state":"VALIDATING"}' https://192.168.17.24/mgmt/tm/transaction/1424388547238937 | python -m json.tool
```

```
{
  "asyncExecution": false,
  "executionTime": 0,
```

```
"executionTimeout": 300,
"failureReason": "",
"kind": "tm:transactionstate",
"selfLink": "https://localhost/mgmt/tm/transaction/1424388547238937?ver=1.0.0",
"state": "COMPLETED",
"timeoutSeconds": 30,
"transId": 1424388547238937
}
```

9. Delete policy iCtrPolicy

TMSH: delete security firewall policy iCtrPolicy

```
[user@server02 workspace]$ curl -sk -u admin:admin -H "Content-Type: application/json" -X DELETE https://192.168.17.24/mgmt/tm/security/firewall/policy/~Common~iCtrPolicy | python -m json.tool
```

No JSON object could be decoded



TIP: Active policies (applied to a context) cannot be deleted until they are detached from the said context. iControl REST is smart enough to explain the reason for request failures in a reply message.

```
[user@server02 workspace]$ curl -sk -u admin:admin -H "Content-Type: application/json" -X DELETE https://192.168.17.24/mgmt/tm/security/firewall/policy/~Common~mypolicy | python -m json.tool
{
  "code": 400,
  "errorStack": [],
  "message": "01070635:3: The policy (/Common/mypolicy) is referenced by one or more firewalled objects."
}
```

IV CONCLUSION

iControl REST API is a powerful tool that enables users to manage AFM configuration objects easily. It is easy to create new REST requests by following the principles that were described in this document. iControl REST is capable of doing anything tmsh is programmed to do.

tmsh can be used to discover all available commands and options for a given component. A reference guide is available at: https://support.f5.com/kb/en-us/products/big-ip_ltm/manuals/product/bigip-tmsh-reference-11-6-0.html

