Failover with iControl part II: The utility.

Don MacVittie, 2007-30-11

Last time we developed a Java tool that will let you query the status of a BIG-IP to determine if it is in a redundant pair, and if so, what its status is. In this article we're going to develop that utility into a command-line utility for bringing BIG-IP boxes up and down by toggling between the members of a redundant pair. The actual work to toggle between two members of a redundant pair is pretty trivial with iControl, there are routines - set_standby() and set_failback() in the System.Failover interface.

The problem comes in when you consider that you really don't want the ability to failover and failback to be arbitrarily on all the time, so the developers put in a bit of a break to make certain you knew what you were doing and you proved you wanted to do it. There is a key that must exist in the database on the BIG-IP that tells that particular BIG-IP it is to honor failover and failback requests submitted from anywhere but the redundant pair.

So to build this little utility, we need to perform several steps:
1. Write a command line parsing routine to allow users to pass commands in (and since we're doing that, you'll find the one we developed takes username, password, and BIG-IP address also). We leave that to the source at the end of the article, you can peruse it at your liesure since it has little to do with failing over the BIG-IP and is really just an excercise in parsing Java command lines.
2. Write a set of routines to query for the database key and its value, and if it doesn't exist, set the value.
3. Write a set of routines to actually put a unit into standby or active.

None of these are a huge deal to implement, and we'll just walk through the two we're covering.

First up is the routines to query for and create the database key. We want them to be generic enough that should we run into a similar problem in the future, we can utilize them again. Toward that end, we wrap the web services calls in much the same manner that we used for many calls in the Java Wrappers Project - very lightly. First, the routine to ask about the DB key...

```
public static boolean dbVarDefined(String varName)
  {
     ManagementDBVariableBindingStub stub;
     boolean ret = false;
     String param[] = {varName};

     try {
        stub = (ManagementDBVariableBindingStub) new
ManagementDBVariableLocator().getManagementDBVariablePort(new URL(fullUrl));
        stub.setTimeout(6000);
        // Make the call, take first element of the return array - since we only sent one varname.
        ret = (stub.is_variable_available(param)[0]);
     } catch (Exception e) {
        System.out.println("dbVarDefined Error: " + e.getLocalizedMessage());
     }
     return ret;
  }
```

Notice that if an exception of any kind is raised by the call, ret will never get set. Thus we have a routine that will tell us if any given database variable key is defined on our BIG-IP. So now we can ask if it exists, the next problem is what to do if it doesn't. For that we have the create method...

```
public static boolean createKey(String varName, String value)
   {
      ManagementDBVariableBindingStub stub;
      ManagementDBVariableVariableNameValue nv[] = {new ManagementDBVariableVariableNameValue()};
      nv[0].setName(varName);
      nv[0].setValue(value);

      try {
         stub = (ManagementDBVariableBindingStub) new
ManagementDBVariableLocator().getManagementDBVariablePort(new URL(fullUrl));
         stub.setTimeout(6000);

         stub.create(nv); // create the name/value combination that is.
      } catch (Exception e) {
         System.out.println("createKey Error: " + e.getLocalizedMessage());
         return false;
      }
      return true;
   }
```

In this case, our default is to return true, so we explicitly return false in the exception handler to make certain word got out.

Okay, so now we have the key that says "this BIG-IP can be failed over manually" - assuming we call these routines. The key (variable) name we need is Failover.ManFailBack and the value we're setting is enable.

Next up are the actual routines that will switch the BIG-IP between active and standby modes. These are regular old iControl calls with not too much to them, so I'll just present them here:

```
/* will take an active device and make it the standby, automatically promoting the other device to the active device. */
   public static void setStandby()
   {
      SystemFailoverBindingStub stub;

      try {
         stub = (SystemFailoverBindingStub) new SystemFailoverLocator().getSystemFailoverPort(new URL(fullUrl));
         stub.setTimeout(6000);
         stub.set_standby();
      } catch (Exception e) {
         System.out.println("isRedundant Error: " + e.getLocalizedMessage());
      }
   }

   /* will take a device in standby and make it the active device if it has failed in the past, automatically promoting the other
device to the active device. */
   public static void setFailback()
   {
      SystemFailoverBindingStub stub;

      try {
         stub = (SystemFailoverBindingStub) new SystemFailoverLocator().getSystemFailoverPort(new URL(fullUrl));
         stub.setTimeout(6000);
         stub.set_failback();
      } catch (Exception e) {
         System.out.println("isRedundant Error: " + e.getLocalizedMessage());
```

```
    }
  }
```

That's all there is to them. The first one sets the device you're communicating with into standby mode (which will make the other go active if they are an active/standby pair), the second brings it back. Note that "failback" is not 100% active, this routine will take any device in standby and make it active... It does not require a previous failure, just that the device be in standby mode.

Now with those four routines, and the information from our previous article, we're ready to build that command line tool. There are some serious changes to the main() routine from last time, so we'll talk a bit about them before I add the source at the end and we call this story done.

First off, since we set up parseArgs to take our authentication information, we need to have it return a boolean value to tell us if all is well.We don't want to try and hook up with a BIG-IP without an address or username, for example.

So in main, we set up an if statement to check the return of parseArgs:

```
if(!parseArgs(args)) {
      System.out.println("Incorrect parameters, exiting.");
      return;
  }
```

Of course, parseArgs() provides more detailed error reporting when it finds problems with the parameters.

Next we wrap all of the important stuff in a call to see if this box is even part of a redundant pair - if not, we should not do anything (you may wish to change this if you have a BIG-IP you want to force into Standby mode that is not part of a redundant pair, but we don't recommend doing that.

```
if(checkRedundant())
```

In the next few lines, we check the current state of the BIG-IP against the request - if you've asked it to "failback", but the machine is in the active state, we tell you that there's a problem, for example.
After we're comfortable with the state, we output some information for the user - the current state of the BIG-IP, and the requested action.

Then we take a few lines to check for, and then set the BIG-IP into the state the user requested.

That's it! You can toggle them back and forth to your heart's content.

The team has told me I'm not allowed to use the formatting style that I've used in these two articles for source code anymore... So I'll try regular old Courier New in the "toggle standby" source code.

USAGE NOTES: The best use of this program is to call the primary with /failover, perform the upgrade, then call the secondary with /failover. While calling with /failback, we encountered some difficulties that we're lookiing into.

```
package f5Sample;

import iControl.SystemFailoverBindingStub;
import iControl.SystemFailoverLocator;
import iControl.SystemFailoverFailoverState;
import iControl.SystemFailoverFailoverMode;
import iControl.ManagementDBVariableBindingStub;
import iControl.ManagementDBVariableLocator;
```

```
import iControl.ManagementDBVariableVariableNameValue;

import java.net.URL;

/**
 * @author dmacvittie
 * @copyright 2007, F5 Networks, Inc.
 *
 */
public class FailoverMain {

    static String username = "", password = "", ipAddress = "", fullUrl;
    static boolean failover = false, failback = false;
    /**
     *
     */
    public FailoverMain() {
        // TODO Auto-generated constructor stub
    }

    /**
     * @param args
     */
    public static void main(String[] args) {
        System.setProperty("javax.net.ssl.trustStore",
System.getProperty("user.home") + "/.keystore");
        XTrustProvider.install();

        if(!parseArgs(args)) {
            System.out.println("Incorrect parameters, exiting.");
            return;
        }

        if(checkRedundant()) {
            // This is part of a redundant pair
            SystemFailoverFailoverState failoverState = checkFailoverState();

 if(failoverState.toString().equals(SystemFailoverFailoverState._FAILOVER_STATE_ACTIVE)
&& failback) {
                System.out.println("Error: System is not in failover state, but
request was to fail back. exiting.");
                return;
            }

 if(failoverState.toString().equals(SystemFailoverFailoverState._FAILOVER_STATE_STANDBY)
&& failover) {
                System.out.println("Error: System is in failover state, but
request was to fail over. exiting.");
                return;
            }

            System.out.println("Current failover state is: " +
failoverState.toString());
```

```
            String foRequest;
            if(failover)
                foRequest = "Failover";
            else  // Assume it's one or the other by this point - parseArgs fails
if neither or both are set.
                foRequest = "Failback";

            System.out.println("Requested: " + foRequest + ".\nAttempting
requested action...");

            /* Now the tricky part... There is a key that must exist and have a
specific value
             * in the Database on the BIG-IP for you to start flicking redundant
pairs back and forth... So let's go looking for it.
             */

            if(!dbVarDefined("Failover.ManFailBack")) {
                System.out.println("Manual failover DB key does not exist.
Creating...");
                if(!createKey("Failover.ManFailBack", "enable")) {
                    System.out.println("Error creating DB Variable for manual
failover. Please check user rights.");
                    return;
                }
            }

            if(failover) {    // Can assume device is in active state or would
have exited.
                setStandby();
            } else {         // Can assume failback is set and device is in
standby, or wouldn't have gotten here.
                setFailback();
            }
        } else
            System.out.println("Your BIG-IP is not part of a redundant pair.
Exiting.");

        System.out.println("Done.");

    }

    public static boolean parseArgs(String args[]) {
        // Parse them out...
        for(int i=0;i<args.length;i++) {
            if(args[i].compareToIgnoreCase("/failover") == 0)
                failover=true;
            else if(args[i].compareToIgnoreCase("/failback") == 0)
                failback=true;
            else if(args[i].toUpperCase().startsWith("/UNAME"))
                username = args[i].substring(7);
            else if(args[i].toUpperCase().startsWith("/PASSWORD"))
                password = args[i].substring(10);
            else if(args[i].toUpperCase().startsWith("/IP"))
```

```
                ipAddress = args[i].substring(4);
            else
                System.out.println("Unrecognized command: " + args[i] + "
ignored.");
        }
        // Check that we have all of our params, and that exactly one of failover
or failback is set.
        if(username.length() == 0 || password.length() == 0 || ipAddress.length()
==0 || (failover == failback)) {
            System.out.println("Usage: \nJava FailoverSample /username=XXXX
/password=YYYY /IP=111.111.111.111 (/failover or /failback)")
            System.out.println("\tEither /failover or /failback may be specified,
not both.");
            return false;
        }
        // Check that there are at least digits in the first character of the IP
address (looking for DNS names).
        if(!Character.isDigit(ipAddress.charAt(0))) {
            System.out.println("Parameter error. This version of Failover does not
resolve host names. Please supply a valid IP address.");
            return false;
        }
        // Bulid the connection string.
        setInfo();

        return true;

    }

    public static void setInfo() {
        fullUrl = "https://" + username + ":" + password + "@" + ipAddress +
"/iControl/iControlPortal.cgi";
    }

    public static boolean dbVarDefined(String varName) {
        ManagementDBVariableBindingStub stub;
        boolean ret = false;
        String param[] = {varName};

        try {
               stub = (ManagementDBVariableBindingStub)
                new ManagementDBVariableLocator().getManagementDBVariablePort(new
URL(fullUrl));

            stub.setTimeout(6000);

            // Make the call, take first element of the return array - since we
only sent one varname.
            ret = (stub.is_variable_available(param)[0]);
        } catch (Exception e) {
            System.out.println("isRedundant Error: " + e.getLocalizedMessage());
        }
        return ret;
```

```
    }
    public static boolean createKey(String varName, String value) {
        ManagementDBVariableBindingStub stub;
        ManagementDBVariableVariableNameValue nv[] = {new
ManagementDBVariableVariableNameValue()};
        nv[0].setName(varName);
        nv[0].setValue(value);
        try {
               stub = (ManagementDBVariableBindingStub)
                new ManagementDBVariableLocator().getManagementDBVariablePort(new
URL(fullUrl));

            stub.setTimeout(6000);

            // Make the call, take first element of the return array - since we
only sent one varname.
            stub.create(nv);
        } catch (Exception e) {
            System.out.println("isRedundant Error: " + e.getLocalizedMessage());
            return false;
        }
        return true;
    }

    public static boolean checkRedundant() {
        SystemFailoverBindingStub stub;
        boolean ret = false;
        try {
               stub = (SystemFailoverBindingStub)
                new SystemFailoverLocator().getSystemFailoverPort(new
URL(fullUrl));

            stub.setTimeout(6000);

            ret = stub.is_redundant();
        } catch (Exception e) {
            System.out.println("isRedundant Error: " + e.getLocalizedMessage());
        }
        return ret;
    }

    /* will take an active device and make it the standby, automatically promoting
the other device to the active device. */
    public static void setStandby() {
        SystemFailoverBindingStub stub;
        try {
               stub = (SystemFailoverBindingStub)
                new SystemFailoverLocator().getSystemFailoverPort(new
URL(fullUrl));

            stub.setTimeout(6000);

            stub.set_standby();
```

```
        } catch (Exception e) {
            System.out.println("isRedundant Error: " + e.getLocalizedMessage());
        }

    }

    /* will take a device in standby and make it the active device if it has
failed in the past, automatically
        promoting the other device to the active device. */
    public static void setFailback() {
        SystemFailoverBindingStub stub;
        try {
               stub = (SystemFailoverBindingStub)
                new SystemFailoverLocator().getSystemFailoverPort(new
URL(fullUrl));

            stub.setTimeout(6000);

            stub.set_failback();
        } catch (Exception e) {
            System.out.println("isRedundant Error: " + e.getLocalizedMessage());
        }

    }

    public static SystemFailoverFailoverState checkFailoverState() {
        SystemFailoverBindingStub stub;
        SystemFailoverFailoverState ret =
SystemFailoverFailoverState.FAILOVER_STATE_ACTIVE;
        try {
               stub = (SystemFailoverBindingStub)
                new SystemFailoverLocator().getSystemFailoverPort(new
URL(fullUrl));

            stub.setTimeout(6000);

            ret = stub.get_failover_state();
        } catch (Exception e) {
            System.out.println("State Error: " + e.getLocalizedMessage());
        }

        return ret;


    }

    public static SystemFailoverFailoverMode checkMode() {
        SystemFailoverBindingStub stub;
        SystemFailoverFailoverMode ret =
SystemFailoverFailoverMode.FAILOVER_MODE_ACTIVE_ACTIVE;
        try {
               stub = (SystemFailoverBindingStub)
                new SystemFailoverLocator().getSystemFailoverPort(new
```

```
URL(fullUrl));

            stub.setTimeout(6000);

            ret = stub.get_failover_mode();
        } catch (Exception e) {
            System.out.println("Mode Error: " + e.getLocalizedMessage());
        }

        return ret;


    }
    public static String findPeer() {
        SystemFailoverBindingStub stub;
        String ret[] = {"No peer found"};
        try {
               stub = (SystemFailoverBindingStub)
                new SystemFailoverLocator().getSystemFailoverPort(new
URL(fullUrl));

            stub.setTimeout(6000);

            ret = stub.get_peer_address();
        } catch (Exception e) {
            System.out.println("Peering Error: " + e.getLocalizedMessage());
        }

        return ret[0];

    }
    public static boolean setFailbackDBVal() {
        boolean ret[] = {false};
        String input[] = {"Failover.ManFailBack"};
        ManagementDBVariableBindingStub stub;
        try {
               stub = (ManagementDBVariableBindingStub)
                new ManagementDBVariableLocator().getManagementDBVariablePort(new
URL(fullUrl));

            stub.setTimeout(6000);

            ret = stub.is_variable_available(input);

            if(ret[0] == false) {
                String values[] = {"enable"};
                ret[0] = createDBVariable(input, values);
            }


        } catch (Exception e) {
            System.out.println("Peering Error: " + e.getLocalizedMessage());
        }
```

```
        return ret[0];
    }

    public static boolean createDBVariable(String names[], String values[]) {
        boolean ret = false;
        ManagementDBVariableVariableNameValue input[] = {new
ManagementDBVariableVariableNameValue()};
        input[0].setName(names[0]);
        input[0].setValue(values[0]);
        ManagementDBVariableBindingStub stub;
        try {
               stub = (ManagementDBVariableBindingStub)
                new ManagementDBVariableLocator().getManagementDBVariablePort(new
URL(fullUrl));

            stub.setTimeout(6000);

            stub.create(input);
            ret = true;

        } catch (Exception e) {
            System.out.println("Peering Error: " + e.getLocalizedMessage());
        }

        return ret;
    }
}
```

F5 Networks, Inc. | 401 Elliot Avenue West, Seattle, WA 98119 | 888-882-4447 | f5.com

F5 Networks, Inc.	F5 Networks	F5 Networks Ltd.	F5 Networks
Corporate Headquarters	Asia-Pacific	Europe/Middle-East/Africa	Japan K.K.
info@f5.com	apacinfo@f5.com	emeainfo@f5.com	f5j-info@f5.com