iC2I: Dumping a Virtual.

Don MacVittie, 2008-10-05

In the quest for iControl mastery, one thing that helps is a mountain of examples. There are a few excellent examples distributed with the API downloads, but more is always better. The i2CI series aims to give you easy-to-use utility functions, but for a short series we've decided to back up and show you some good general iControl programming points. This installment expands the VirtualServer class by adding a "Dump" routine that will write out the core information about a Virtual to standard out. The routine isn't rocket science, but it offers you examples of how the various getXXX routines are called. Future installments will take some of the learning from this article and apply them to building a tree view of the BIG-IP.

First we'll offer up the source. This method can be cut and pasted directly into the BigIpVirtualServer class that we've defined in previous installments of i2CI. All it does is take the name of a Virtual Server and dump key information about that Virtual to the screen. With that said, there are a few tricky bits that we'll talk about after the source.

If you are using Eclipse with iControl included in your project, then cutting and pasting the dump() routine will automatically insert the required *import* statements. If you're using another editor, you'll need the following lines in the imports section of the VirtualServer class:

```
import iControl.LocalLBVirtualServerVirtualServerResource;
 import iControl.LocalLBVirtualServerVirtualServerProfile;
 import iControl.LocalLBVirtualServerVirtualServerType;
 import iControl.CommonProtocolType;
 import iControl.LocalLBHardwareAccelerationMode;
 import iControl.CommonULong64;
 import iControl.CommonStatistic;
 import iControl.CommonTimeStamp;
 import iControl.LocalLBVirtualServerVirtualServerType;
 import iControl.CommonEnabledState;
 import iControl.CommonIPPortDefinition;
 import iControl.LocalLBObjectStatus;
 import iControl.LocalLBVirtualServerVirtualServerProfileAttribute;
 import iControl.LocalLBVirtualServerVirtualServerRule;
 import iControl.CommonVLANFilterList;
 import iControl.LocalLBVirtualServerVirtualServerStatistics;
 import iControl.LocalLBVirtualServerVirtualServerStatisticEntry;
```

The rest of the code utilizes the items above to show you the status of the Virtual you pass in.

```
    // Dump will write out important and commonly used information to the command line for debugging
     public void dump(String virtualName) throws java.lang.Exception {
         String virtList[] = {virtualName};

         // Print out the header
         System.out.println("Virtual " + virtualName + " Informational Dump.");
         System.out.println();

         // Get the pool list.
```

```
        String pools[] = getDefaultPools(virtList);
        System.out.println("Pools this Virtual relies upon:");
        for(int i = 0; i < pools.length; i++)
            System.out.println("\t" + pools[i]);
        System.out.println();

        // Get Hardware Acceleration.
        LocalLBHardwareAccelerationMode hwAccel[] = stub.get_actual_hardware_acceleration(virtList);
        System.out.println("Acceleration Mode: " + hwAccel[0]);
        System.out.println();

        // Get connection limits and connection mirroring.
        CommonULong64 limits[] = stub.get_connection_limit(virtList);
        UsefulU64 uLimit = new UsefulU64(limits[0]);

        CommonEnabledState mirrorState[] = stub.get_connection_mirror_state(virtList);

        System.out.println("Maximum connections: " + uLimit.toString());
        if(mirrorState[0] == CommonEnabledState.STATE_DISABLED)
            System.out.println("Mirroring is enabled");
        else
            System.out.println("Mirroring is disabled");
        System.out.println();

        // Get the Virtual Server Type.
        LocalLBVirtualServerVirtualServerType vsType[] = stub.get_type(virtList);
        System.out.println("Virtual Server Type: " + vsType[0].toString());
        System.out.println();

        // Get the IP Address, mask, and protocol.
        CommonIPPortDefinition cip[] = stub.get_destination(virtList);
        System.out.println("Virtual is advertised as: ");
        System.out.println("\tIP:\t\t" + cip[0].getAddress());
        System.out.println("\tPort:\t\t" + cip[0].getPort());

        System.out.println("\tNetmask:\t" + stub.get_wildmask(virtList)[0]);
        System.out.println();

        // Get State and Status.
        CommonEnabledState vsEnabledState[] = stub.get_enabled_state(virtList);
        if(vsEnabledState[0] == CommonEnabledState.STATE_ENABLED)
            System.out.println("Virtual is enabled.");
        else
            System.out.println("Virtual is disabled.");

        LocalLBObjectStatus vsObjStatus[] = stub.get_object_status(virtList);
        System.out.println("Object status is: " + vsObjStatus[0].getStatus_description());
        System.out.println();

        // Get Profiles.
        LocalLBVirtualServerVirtualServerProfileAttribute profs[][] = stub.get_profile(virtList);
        System.out.println("Profile Information");
        for(int i = 0; i < profs[0].length; i++) {
            System.out.print("\tProfile " + profs[0][i].getProfile_name());
            System.out.print(" of type " + profs[0][i].getProfile_type().toString());
            System.out.println(" is applied in context: " + profs[0][i].getProfile_context().toStrin
        }
```

```
        System.out.println();

        // Get Rules.
        LocalLBVirtualServerVirtualServerRule vsRules[][] = stub.get_rule(virtList);
        System.out.println("Rules: ");
        for(int i=0; i <vsRules[0].length; i++)
            System.out.println("\tThe rule named " + vsRules[0][i].getRule_name() + " is running wit
        System.out.println();

        // Get the VLANs it listens on.
        CommonVLANFilterList vlanFilters[] = stub.get_vlan(virtList);
        for(int i = 0; i < vlanFilters.length; i++) {
            System.out.print("The following VLANs are ");
            if(vlanFilters[i].getState() == CommonEnabledState.STATE_DISABLED)
                System.out.println("disabled.");
            else
                System.out.println("enabled.");
            String nameList[] = vlanFilters[i].getVlans();

            for(int j = 0; j < nameList.length; j++)
                System.out.println("\t" + nameList[j]);
            if(nameList.length == 0)
                System.out.println("\tEmpty list.");
            System.out.println();
        }
        // Get select statistics.
        LocalLBVirtualServerVirtualServerStatistics stats = stub.get_statistics(virtList);
        CommonTimeStamp time = stats.getTime_stamp();
        System.out.println("Statistics as of " + time.getHour() + ":" + time.getMinute() + "." + tim
                " on " + time.getMonth() + "/" + time.getDay() + "/" + time.getYear());
        LocalLBVirtualServerVirtualServerStatisticEntry statAggregate[] = stats.getStatistics();
        for(int j = 0; j < statAggregate.length; j++ ) {
            CommonStatistic statDetail[] = statAggregate[j].getStatistics();
            for(int i = 0; i < statDetail.length; i++) {
                UsefulU64 uStat = new UsefulU64(statDetail[i].getValue());
                System.out.println("\t" + statDetail[i].getType().toString() + "\t" + uStat.toString
            }
        }
    }
```

The first interesting point is to note that we're using the Useful64 class we defined in a previous installment as a worker class - many of the data values BIG-IP returns are returned in 64 bit data values split into two 32 bit values, and Useful64 simply makes it easier to deal with them - particularly with formatting and printing them, but also with converting the values to 64 bit integers.

Next up is the fact that many iControl calls return nested arrays. The first dimension of these arrays is always the virtuals passed in, the second dimension is the elements that apply to that virtual. Since we're passing a single Virtual name in, this routine always takes array[0][i] from these return values, where [i] is the list applicable to the virtual name we passed in.

Note that the pattern is repeatable and the same for these routines - declare a variable of the appropriate type to hold the return value, call a stub routine, then process the return value. That's true of all of the get_xxx() routines in iControl, assuming you already have the stub declared (as we do because the connection is made in the constructor).

For statistics note the nested objects with the following ownership:

LocalLBVirtualServerVirtualServerStatistics is the top level element. It holds the timestamp and an array of LocalLBVirtualServerVirtualServerStatisticEntry elements that represent the statistics lists.

LocalLBVirtualServerVirtualServerStatisticEntry holds an array of elements of type CommonStatistic

CommonStatistic holds the statistic name and the current 64 bit value. (there's a timestamp field, but the wiki entry for CommonStatistic mentions it as not yet supported, and we have the timestamp of the batch to use).

So we have to run through a couple of nested loops to get the statistics values, but other than that there is no rocket science to these items either.

That's about it for items with special handling. This routine will not be the basis for the work we do in the next couple of installments, but many of the calls used in this article will be re-used as we build a BIG-IP treeview.

Get the Flash Player to see this player.

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

F5 Networks, Inc.
Corporate Headquarters
info@f5.com

F5 Networks
Asia-Pacific
apacinfo@f5.com

F5 Networks Ltd.
Europe/Middle-East/Africa
emeainfo@f5.com

F5 Networks
Japan K.K.
f5j-info@f5.com