iControl 101 - #19 - Time Conversions

Joe Pruitt, 2008-10-07

In the iControl API there are several methods that return time values. Some are in nicely packed structures with the year, month, day, hour, minute, and second extracted for you, but there are other methods that return just a single long integer for the time. This article will discuss how to work with the different time values and convert them to local time.

Initialization

This article uses PowerShell and the iControl Cmdlets for PowerShell as the client environment for querying the data. The following setup will be required for the examples contained in this article.

```
PS C:\> Add-PSSnapIn iControlSnapIn
PS C:\> Initialize-F5.iControl -Hostname bigip_address -Username bigip_user -Password bigip_pass
```

System Time

To get the system time of your BIG-IP, the System.SystemInfo interface has the get_time() and get_time_zone() methods.

```
struct TimeStamp {
  long year;
  long month;
  long day;
  long hour;
  long minute;
  long second;
};

TimeStamp System.SystemInfo.get_time();

struct TimeZone {
  long gmt_offset;
  String time_zone;
  boolean is_daylight_savings_time;
};

TimeZone System.SystemInfo.get_time_zone();
```

So all you have to do is simply call the get_time() method and you can easily extract the values from the returned structure to build a native time object. In this case we are converting the TimeStamp structure into a .NET DateTime object. Notice that a newly created DateTime object is dated January 1, 0001 because a date with the year 0000 is not valid. When converting the TimeStamp structure, you'll need to subtract a year to account for this.

```
PS C:\> $SystemTime = (Get-F5.iControl).SystemSystemInfo.get_time()
PS C:\> $SystemTime
year   : 2008
month  : 7
day    : 10
hour   : 14
minute : 15
second : 10

PS C:\> $DT = new-object -typename System.DateTime
```

```
PS C:\> $DT = new-object -typename System.DateTime
Monday, January 01, 0001 12:00:00 AM

PS C:\> $DT.AddYears($SystemTime.year-1).AddMonths($SystemTime.month).
AddDays($SystemTime.day).AddHours($SystemTime.hour).
AddMinutes($SystemTime.minute).AddSeconds($SystemTime.second)

Tuesday, August 11, 2008 2:15:10 PM
```

The time returned here is in universal time referred to UTC or GMT. To turn this value into a local time, you need to get the timezone information. The local timezone information for the BIG-IP can be retrieved from the get_time_zone() method. By adding the gmt_offset in the TimeZone structure to the number of hours in the time value, you'll get the local system time.

```
PS C:\> $TimeZone = (Get-F5.iControl).SystemSystemInfo.get_time_zone()
PS C:\> $TimeZone
                            gmt_offset time_zone                                                  is_da
                            ---------- ---------                                                  -----
                                    -7 PDT

PS C:\> $DT = $DT.AddHours($TimeZone.gmt_offset)
PS C:\> $DT

Monday, August 11, 2008 7:15:10 AM
```

The Epoch

Now that you've got the Common.TimeStamp structure converted into a local time, let's move on to the other time values returned in the iControl methods. Several methods return a time value as represented by a long integer. The LocalLB.RAMCacheInformation interface has the LocalLB.RAMCacheInformation.RAMCacheEntry structure that has several long values for received, last_sent, and expiration that are longs and defined in the docs as the number of seconds since the Unix Epoch. There are other methods that return long values that are seconds of time since the epoch such as the Management.LicenseAdministration's get_evaluation_license_expiration() method. Unforutuantely, not all of the documentation for the methods specifically state that time time is the number of seconds since the epoch. But, you can be confident if a method says it returns a time and it's represented as a long it is the number of seconds since the epoch.

For you non-unix heads out there, the word "epoch" is defined as an "instant in time chosen as the origin of a particular era". In unix terms, this epoch is defined as 00:00:00 UTC on January 1, 1970. So, when you hear mention of "Unix Time" or "POSIX Time" it is generally referred to as the number of seconds elapsed since midnight UTC of January 1, 1970.

With that information in hand, it is fairly straightforward to convert it to Universal time and then to local time. In fact, it's easier to convert than the Common.TimeStamp structure above because all you have to do is add the value of the time to the unix epoch as seconds. This can be done with the following code:

```
PS C:\> $epoch = [DateTime]::Parse("Jan 1, 1970 12:00:00AM")
PS> $curtime = (Get-F5.iControl).ManagementLicenseAdministration.get_evaluation_license_expiration().
PS C:\> $curtime
1215702223

PS C:\> $systime = $epoch.AddSeconds($curtime).AddHours($TimeZone.gmt_offset)
PS C:\> $systime
Thursday, May 08, 2008 2:51:09 PM
```

Conclusion

The examples in the above article was written in Windows PowerShell so if you are a .NET user this should have your covered. For you other folks in Java, perl, python or whatever, there should be equivalent objects in your langauage that you can translate the logic in this article into.

Get the Flash Player to see this player.
20080710-iControl101_19_TimeConversions.mp3

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