RateLimit Header Fields

Presented by Darrel Miller

IETF 122 Bangkok

draft-ietf-httpapi-ratelimit-headers-09

Changes

- Added more callouts to the use of Structured Fields including adding SF term definitions
- Window parameter MUST be non-negative, **non-zero**
- Remaining (r)parameter is REQUIRED on RateLimit field

Problem Types – Quota Exceeded

```
HTTP/1.1 429 Bad Request
Content-Type: application/problem+json

{
  "type": "https://iana.org/assignments/http-problem-types#quota-exceeded",
  "title": "Request cannot be satisifed as assigned quota has been exceeded",
  "violated-policies": ["daily","bandwidth"]
}
```

Problem Types – Temporary Reduced Capacity

```
HTTP/1.1 503 Server Unavailable
Content-Type: application/problem+json

{
  "type": "https://iana.org/assignments/http-problem-types#temporary-reduced-capacity",
  "title": "Request cannot be satisfied due to temporary server capacity constraints",
  "violated-policies": ["hourly"]
}
```

Problem Types – Abnormal Usage Detected

```
HTTP/1.1 429 Too Many Requests
Content-Type: application/problem+json

{
  "type": "https://iana.org/assignments/http-problem-types#abnormal-usage-detected",
  "title": "Request not satisfied due to detection of abnormal request pattern",
  "violated-policies": ["hourly"]
}
```

Additional proposed Quota Unit : cost

requests:

This value indicates the quota is based on the number of requests processed by the resource server. Whether a specific request actually consumes a quota unit is implementation-specific.

content-bytes:

This value indicates the quota is based on the number of content bytes processed by the resource server.

concurrent-requests:

This value indicates the quota is based on the number of concurrent requests processed by the resource server.

cost:

This value indicates the quota is based on the cost of a specific request determined by an algorithm defined by the server.

Implementation

```
[Fact]
0 references
public void ParseRateLimitPolicyField()
{
    var rateLimiter = new RateLimiter.RateLimiter();
    rateLimiter.ProcessPolicyField("\"policy1\";q=100");
    rateLimiter.ProcessServiceLimitField("\"policy1\";r=0");
    PolicyCost[] costs = rateLimiter.CalculatePolicyCost(new HttpRequestMessage());
    Assert.Single(rateLimiter.FindInsufficientLimits(costs));
    Assert.Equal(100,rateLimiter.Policies["policy1"].Quota);
}
```

https://github.com/darrelmiller/ratelimiting

Rate limiting with Partition Key

```
[Fact]
0 references
public void PartitionKey() {
        var rateLimiter = new RateLimiter.RateLimiter((request) => request.Method.ToString());
        var readPartitionKey = Convert.ToBase64String(Encoding.UTF8.GetBytes(HttpMethod.Get.ToString()));
        var writePartitionKey = Convert.ToBase64String(Encoding.UTF8.GetBytes(HttpMethod.Post.ToString()));
        rateLimiter.ProcessServiceLimitField($"\"read\";r=1;pk=:{readPartitionKey}:,\"write\";r=1;pk=:{writePartitionKey}:");

        PolicyCost[] readCosts = rateLimiter.CalculatePolicyCost(new HttpRequestMessage());
        Assert.Empty(rateLimiter.FindInsufficientLimits(readCosts));
        rateLimiter.ApplyCost(readCosts);

        PolicyCost[] writeCosts = rateLimiter.CalculatePolicyCost(new HttpRequestMessage() { Method = HttpMethod.Post });
        Assert.Empty(rateLimiter.FindInsufficientLimits(writeCosts));
        rateLimiter.ApplyCost(writeCosts);

        Assert.Single(rateLimiter.FindInsufficientLimits(readCosts));  // Failed 2nd read
        rateLimiter.ApplyCost(readCosts);
}
```