iScope

Author: Saveen Reddy

Date: 6/3/2014

# Introduction

## Motivation

Traditionally, getting at data stored in Cosmos has been a batch-oriented affair. A job is submitted and then some unpredictable time later the results of the job are available. Those results are then downloaded and consumed.

We observe, however, that our users are increasingly trying to use Cosmos to satisfy an *ad hoc* analytic scenario. They are looking to perform quick, highly-aggregated queries on big data and desire to get the results back in seconds.

**iScope** makes this possible.

In this document we’ll start with a brief explanation about iScope scenarios, then walk through using iScope via Scope Studio, and then finally go into the details.

## Overview of iScope

**iScope** allows users to query Cosmos Data and retrieve results in seconds if those queries meet a very specific set of criteria

* The input data MUST be in a structured stream.
* The iScope query should be interactive friendly (i.e. not very execution intensive).

## Examples of Typical Queries that Meet the Criteria

### Simple key/value lookups on a structured stream

wiki = SSTREAM @"/shares/cosmosAdmin/iScopeCosting/WikipediaWordCount.ss";

rs = SELECT \*

FROM wiki

WHERE keyword == "bing";

OUTPUT rs TO CONSOLE;

### Find the frequency of the TOP 100 query terms

e = SSTREAM @"/shares/cosmosAdmin/iScopeCosting/KeyRange.ss";

g = SELECT TOP 100 \*

FROM e;

OUTPUT g TO CONSOLE;

### Finding values between certain key range

e = SSTREAM @"/shares/cosmosAdmin/iScopeCosting/KeyRange.ss";

g = SELECT BucketID, COUNT() AS total\_count

FROM e

WHERE SourceType > 50 AND SourceType <= 200000000000000

GROUP BY BucketID

ORDER BY BucketID;

OUTPUT g TO CONSOLE;

# Using iScope via Scope Studio

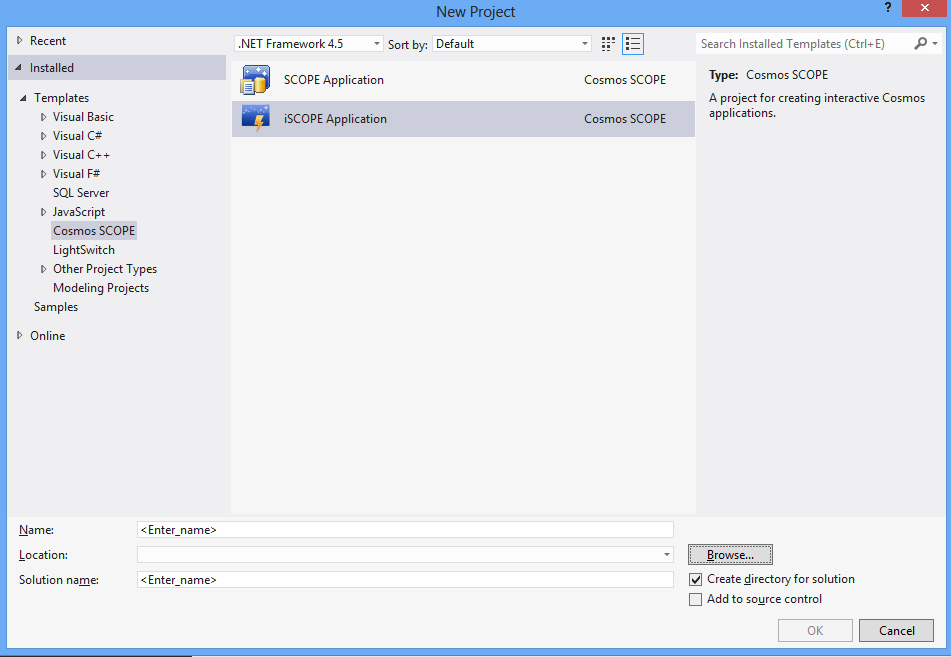
Once you have installed Scope Studio, using iScope is very simple and works like what you are used to with Scope.

The process is simple:

* Create an iScope project.
* Write an iScope script.
* Press **Execute** to run that script.
* View the results.

## Walkthrough

Once you have installed Scope Studio, go to **File > New Project** and in the **New Project** dialog select **Cosmos Scope > iScope Application**.



This will create a new iScope project. You’ll see a new document called **iScope.script** has been created.

Enter the following script:

e = SSTREAM @"/shares/cosmosAdmin/iScopeCosting/KeyRange.ss";

g = SELECT URL, BucketID, SourceType, Market

FROM e

WHERE URL == "1cc961f205e04d18b0c80e34b72c25f5c3a17648ed49443b902d841fba8f5a6b3db040766c984d6fb1e205de0db0e06a06" AND

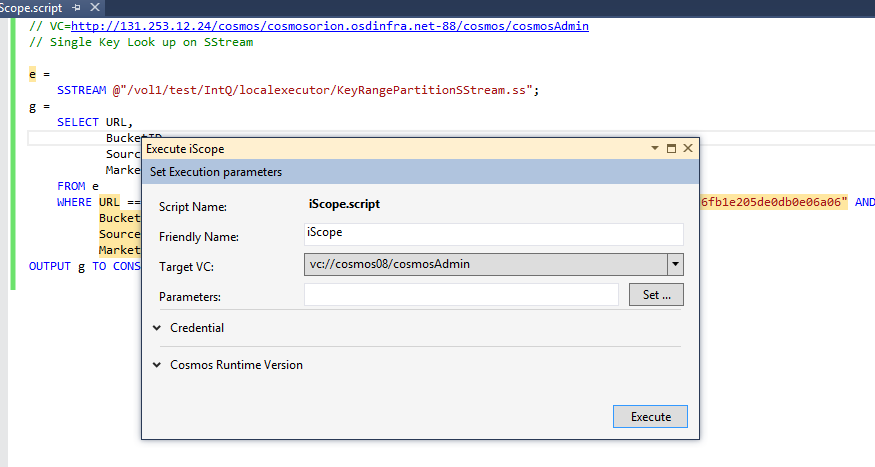
BucketID == 1839513836 AND

SourceType == 1458350729 AND

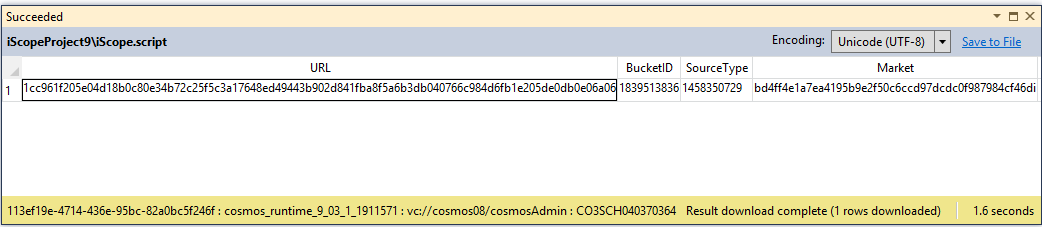
Market == "bd4ff4e1a7ea4195b9e2f50c6ccd97dcdc0f987984cf46di";

OUTPUT g TO CONSOLE;

Press the **Execute** button in the toolbar. This will launch the **Execute iScope** dialog – just press **Execute**.

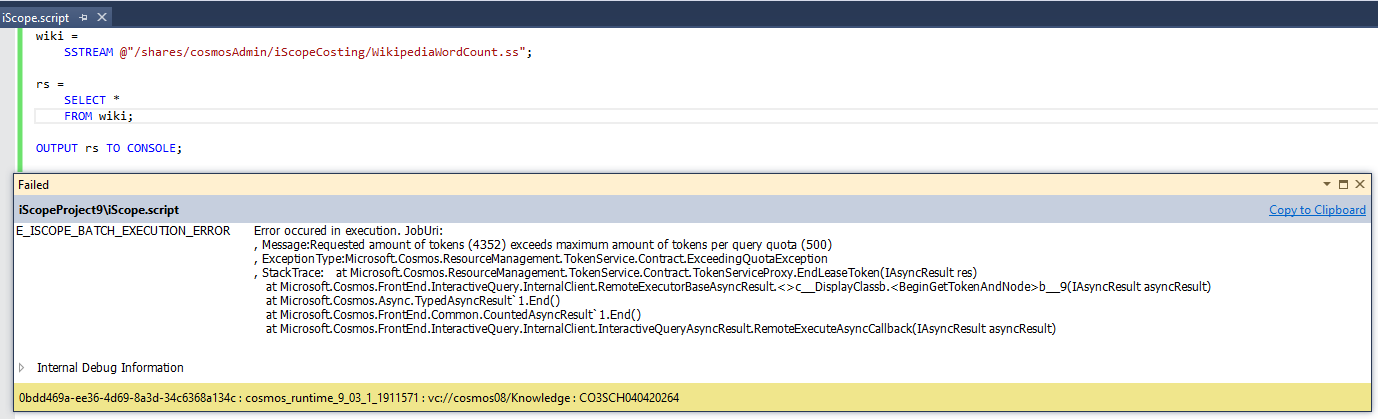


If all goes well, then you will see a single row in the results window.



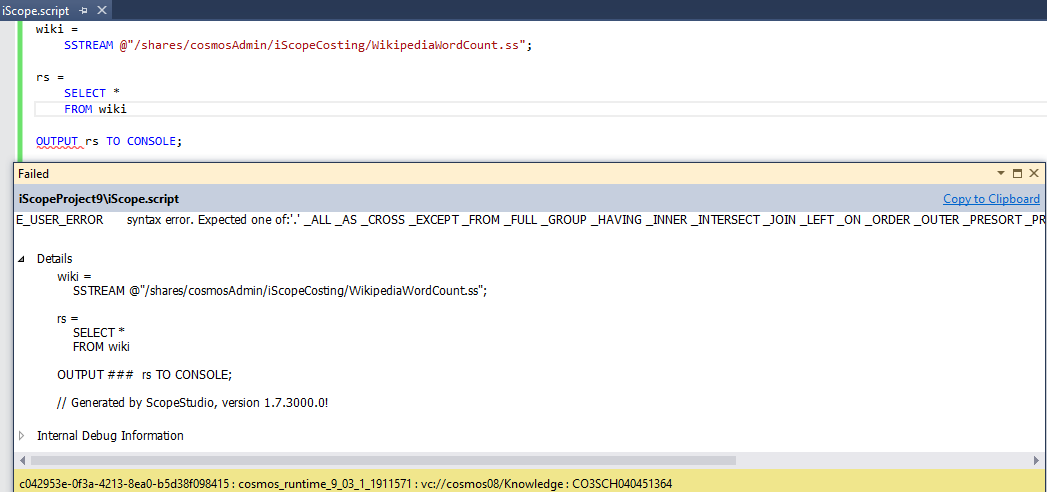
## Queries that are Too Expensive

We’ll cover understanding the cost of an iScope query later on, but you should know what it looks like now. Below is an example:



## Compilation Failures

A query that failed due to compilation errors will look like this:



# iScope Feature Support

Now that we’ve run through the basic usage of iScope, let’s dig deeper in the features iScope provides – especially in comparison to traditional Scope.

## iScope versus Scope

The iScope language is a strict subset of the Scope language.

## Operators

Below is a comprehensive list of all the operators that iScope supports compared to scope. Those operators that are unsupported are shown with strike-through ( ~~EXAMPLE~~ ).

### Common Operators

SELECT

WHERE

HAVING

ORDER BY

### Input and Output

OUTPUT TO CONSOLE

### Aggregations

GROUP BY

AVG

SUM

COUNT

COUNTIF

MAX

MIN

TOP

FIRST

LAST

JOINS

~~ARGMAX~~

~~LIST~~

~~STDDEV~~

### Logical Operators

x AND y

x OR y

NOT x

~~x && y~~

~~x || y~~

~~!x~~

### Comparison Operators

x == y

x > y

x >= y

x <= y

x < y

### Bitwise Operators

x >> y

x <<y

~x

x ^ y

### Mathematical

x \* y

x / y

x + y

x – y

-x

+x

### Other Operators

VIEW

table.column

x AS y

(x)

x % y

x ?? y

(x) y // cast

## Native Types

Here’s the list of Scope types for which there will be full native support. In general, iScope supports all arithmetic operations on these types, i.e. unary and binary operators. All comparison operators are supported, where C# supports it. New instances of these types can be created if and only if they allow literal constants.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | C# Type | Support | | Bool | Full | | bool? | All nullable types have HasValue and Value properties supported. | | Sbyte | Full | | sbyte? | Full | | Char | Full | | Byte | Full | | byte? | Full | | Short | Full | | short? | Full | | Ushort | Full | | ushort? | Full | | Int | Full | | int? | Full | | Uint | Full | | uint? | Full | | Long | Full | | long? | Full | | Ulong | Full | | ulong? | Full | | float | Full | | float? | Full | | Double | Full | | double? | Full | | Decimal | Full | | decimal? | Full | | String | + is not supported, isNullOrEmpy, Empty are supported. | | Guid | Full | |  |  | |

## OUTPUT Statements

Like Scope, all iScope scripts must have an **OUTPUT** statement.

Indeed, iScope must have exactly one **OUTPUT** statement and it must always direct the output to **CONSOLE**.

OUTPUT data to CONSOLE;

# Unsupported Features

## Local Execution

Unlike batch Scope, iScope does not support local execution on your box. To run iScope you must run it on a cluster using your Cosmos credentials and VC name.

## Methods on Types

Scope users are used to calling member methods on their types. For example”:

rs = SELECT a, b.StartsWith(“Foo”) AS c

FROM data;

In the above example, **StartsWith()** is one of the intrinsic methods defined on the .NET string type.

**iScope** does NOT support calling any such intrinsic methods.

In the future, we intend to selectively relax this restriction.

## Other Explicitly Unsupported Keywords

* REGEX support
* REFERENCE
* RESOURCE
* USING
* #DECLARE
* UNION ALL
* IMPORT
* SSTREAM USING

## Other Explicitly Unsupported Features

* Optimization hints/plan
* Unstructured streams
* User-defined functions
* User-defined types
* User-defined objects (reducer, processors, combiners etc.)

# Costing

Using **iScope** requires capacity in the Cosmos system and the unit of this capacity is called a **microtoken**.

## Summary

To execute an iScope query costs a number of **microtokens**. The exact number depends on the queries usage of disk I/O, memory, and CPU cycles needed to execute the query.

Each **virtual cluster** is allocated a fixed number of microtokens. This allocation is determined by the administrator of the virtual cluster in cooperation with the Cosmos team.

If a user submits an **iScope** query for execution, it will only start executing if the cost of the query is less than or equal to the number of microtokens available in that virtual cluster. If not enough microtokens are available, then the iScope query will not be executed and an error will be returned immediately.

## Estimating iScope Query Cost

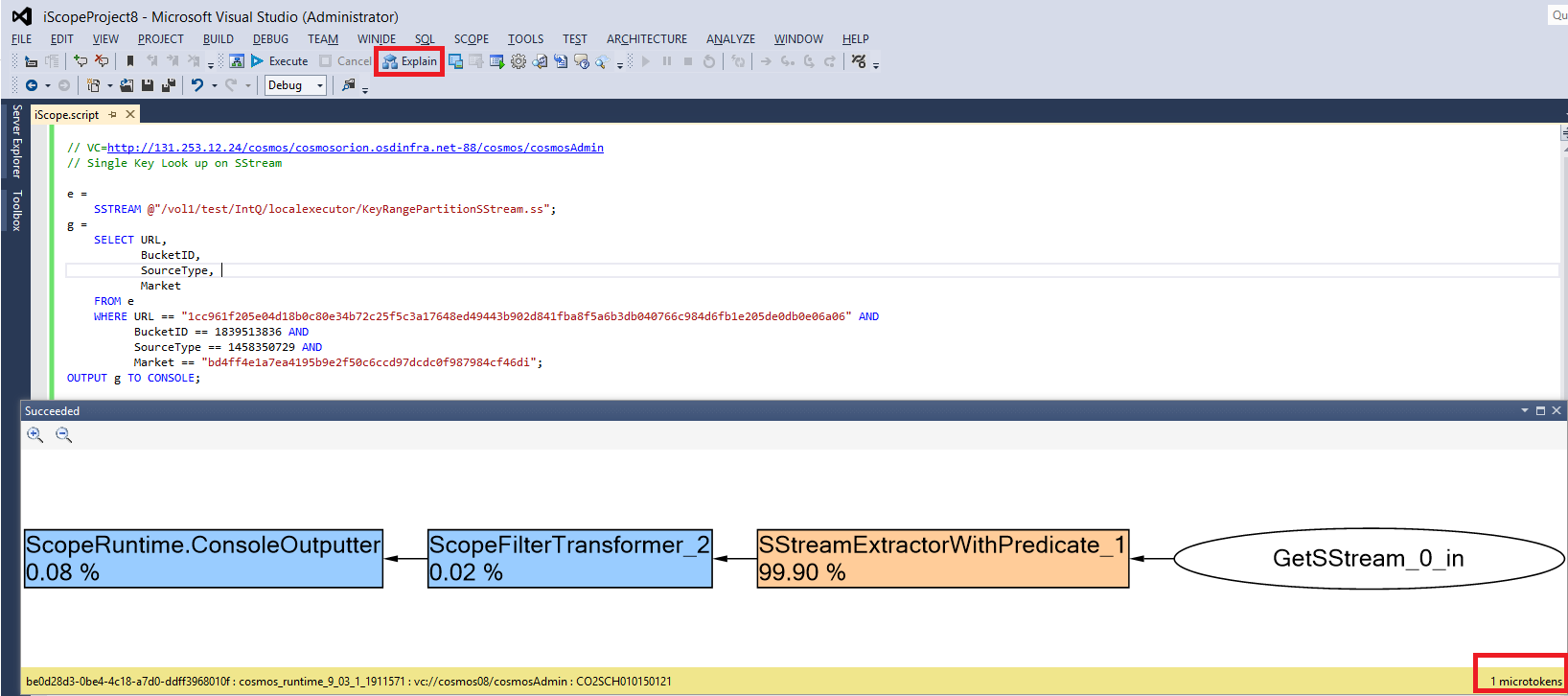
Cosmos lets you **estimate** the cost in microtokens of an iScope query. The ability to see this cost is important because it lets you:

* See why a query was not executed.
* Tweak the query to reduce the cost.

Estimation can be done in two ways:

* Programmatically with the ScopeCompiler.Compiler() method in the Microsoft.Cosmos.Client DLL.
* Inside Scope Studio (screenshot below)

**Screenshot: iScope costing in Scope Studio**



## Relation to Scope

The nature of capacity usage in **iScope** and **Scope** are very different.

Despite the name, **microtokens** are not related to traditional **tokens** in any way. Token capacity and Microtoken capacity are independent. Using capacity in one system has no effect on the other.

## Further Reading

For a detailed explanation on iScope costing, refer to the document here (link TBD).

# Rollout in June 2013

## Feature Availability

All Virtual Clusters will be automatically configured for using iScope by default.

## Microtoken Allocation

* Each Virtual cluster will get a certain number of microtokens depending on their Scope usage.
* If you need additional microtokens please talk to your VCAdmin who can then work with the Cosmos team.

# Using iScope via the Command Line

## From the command-line via Cosmos Powershell

The general form for executing an iScope query from the command line is as follows:

$data1= Invoke-CosmosIScopeQuery -QueryString query -VirtualCluster vc

A more complete example is shown below:

Import-Module CosmosPS

$data1= Invoke-CosmosIScopeQuery -QueryString query -VirtualCluster vc

## From the command-line via SCOPE.EXE

The SCOPE.EXE tool in the Cosmos SDK does NOT support executing iScope queries. Use Cosmos PowerShell instead.

# Notes

## Cost Limits

Only queries that cost less than that configured by your VCAdmin can be run.

## Query Time Limits

The iScope system also ensures that any query that takes a longer amount of time is cancelled so that it does not use up all the system resources.

## Amount of Data in Results

There are no limits on the amount of data that can be transferred.

However, queries that transfer a lot of data will see a higher microtokens cost because use more disk i/o.

# Programmatically Issuing an iScope Query

# Relationship to iQuery

* iScope will deprecate iQuery and plans are to decommission iQuery by end of 2013.
* All customers who have not migrated to iScope by then will be given advance notice.

# Future Plans

Future versions of iScope will support Joins, StreamSets and further improvements to latencies by using enhanced backend parallelism.