Building the most efficient device selector query that selects no devices

devblogs.microsoft.com/oldnewthing/20240515-00

May 15, 2024

Raymond Chen

I had an occasion to write a device selector query that selects no devices. The idea is that you have a function that returns a device selector query for “Find all devices that can X”, but you happen to know that X can’t be done because of some unmet prerequisite unrelated to the devices themselves. What’s the most efficient device selector query that selects no devices?

One thing that doesn’t work is returning the empty string. In Advanced Query Syntax, the empty string is the opposite of what we want: The empty string selects *everything*.

Each clause in Advanced Query Syntax takes the form of (property) (match operator) (value). For example,

```
System.Devices.InterfaceClassGuid:="{DEEBE6AD-9E01-47E2-A3B2-A66AA2C036C9}"
System.Size:>1kb
System.FileName:~<"Copy of"
```

One way to specify a query that matches no devices is to specify contradictory requirements.

```
System.Devices.InterfaceEnabled:=System.StructuredQueryType.Boolean#True AND
    System.Devices.InterfaceEnabled:=System.StructuredQueryType.Boolean#False
```

The InterfaceEnabled property cannot be both True and False, so this will match no devices.

On the other hand, I wondered whether this would make two passes, one to gather all the devices where the interface is enabled, and then (to process the AND clause) a second pass over those devices to find the ones where the interface is disabled. Should I add a garbage query at the front to reduce the search space as quickly as possible?

```
System.Devices.InterfaceClassGuid:="{91F9F631-CD92-42C5-91A4-FBAEAF4DBF09}" AND
    System.Devices.InterfaceEnabled:=System.StructuredQueryType.Boolean#True AND
    System.Devices.InterfaceEnabled:=System.StructuredQueryType.Boolean#False
```

The interface class GUID I picked is one that I generated just now, so I know it won’t match any valid interface GUID.

It turns out that this garbage query at the front does help.

The device selector query evaluator contains optimizations if the query includes an AND clause that does an equality comparison against one of the following properties:

	Optimized for DeviceInformationKind	
Property	**DeviceInterface**	**Device**
System.Devices.DeviceInstanceId	Yes	
System.Devices.ClassGuid	Yes	Yes
System.Devices.ContainerId	Yes	Yes
System.Devices.PanelId		Yes

There are shortcuts in the device management subsystem that can find all devices that match one of those properties, so if one of those properties is involved in the query as an AND clause, the query evaluator can use those shortcuts to get the list of candidate devices quickly, rather than having to enumerate all the devices. Once it gets the list of candidates, it can then evaluate the other AND conditions to filter the results.

Therefore, a fast query that produces no results could be

```
System.Devices.DeviceInstanceId:="{91F9F631-CD92-42C5-91A4-FBAEAF4DBF09}" AND
    System.Devices.InterfaceEnabled:=System.StructuredQueryType.Boolean#True AND
    System.Devices.InterfaceEnabled:=System.StructuredQueryType.Boolean#False
```

We filter against one of the magic optimized properties (the device instance ID), which will quickly produce no results. The last two clauses (looking for something that is both true and false) ensure that even if something by some freak of bad luck happens to match our GUID, it will nevertheless be rejected.