Azure Cosmos DB support for Spring Data source code

CAST supports **Azure Cosmos DB** via its NoSQL for Java extension. Details about the support provided for **Java source code** is explained below.

Supported Client Libraries

Azure Cosmos DB Java with Spring SDK	com.microsoft.azure.spring.data.cosmosdb.repository.DocumentDbRepository com.microsoft.azure.spring.data.cosmosdb.repository.CosmosRepository org.springframework.data.repository.CrudRepository	✓
Azure Cosmos DB Async Java with Spring SDK	com.microsoft.azure.spring.data.cosmosdb.repository.ReactiveCosmosRepository org.springframework.data.repository.reactive.ReactiveCrudRepository	✓

Supported Operations

Operations	Method Supported
Insert	• save • saveAll
Select	• existsById • findById • findAll • findAllById • count
Delete	• delete • deleteAll • deleteById • deleteAllById

Above mentioned methods are supported for both Synchronous and Asynchronous clients.

Objects

Icon	Description

	Java CosmosDB Database
	Java CosmosDB Collection
	Java Unknown CosmosDB Database
	Java Unknown CosmosDB Collection

Links

Links are created for transaction and function point needs:

Link type	Source and destination of link	Methods Supported
parentLink	Between CosmosDB Database object and CosmosDB Collection	
useInsertLink	Between the caller Java Method objects and CosmosDB Collection	• save • saveAll
useSelectLink		• existsById • findById • findAll • findAllById • count
useDeleteLink		• delete • deleteAll • deleteById • deleteAllById

What results can you expect?

Once the analysis/snapshot generation is completed, you can view the results in the normal manner (for example via CAST Enlighten). Some examples are shown below.

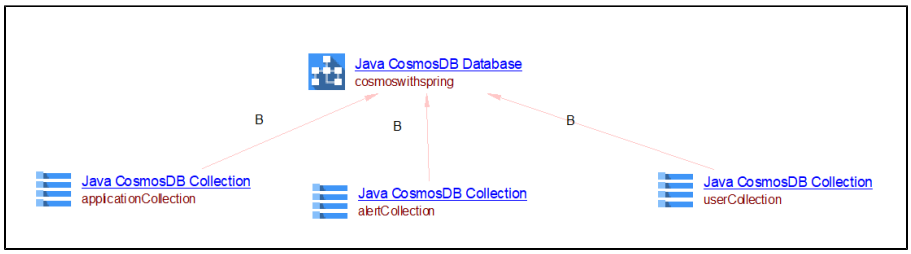
CosmosDB Database Connection from Properties File

application.properties

```
azure.cosmosdb.uri=https://cosmoswithspring.documents.azure.com:443/
azure.cosmosdb.key=2VanbwRYNeJKCUjb5StsnBKaatDcKGRUckHnlSUVSJXifWL3exjwjP2o6rJpt8fY24MRmtnctzcE8hWAlBsRZw==
azure.cosmosdb.database=cosmoswithspring
```

Cosmos DB Client Creation

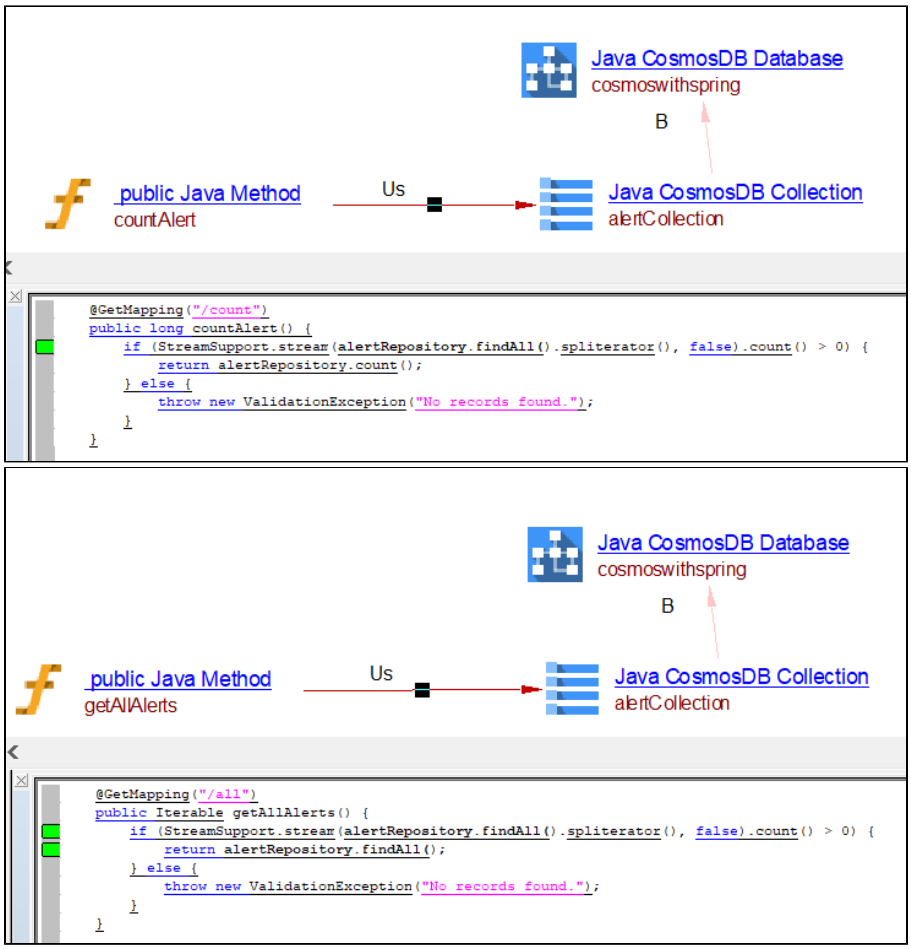
```
@Configuration
@EnableDocumentDbRepositories
public class AppConfig extends AbstractDocumentDbConfiguration {

    @Value("${azure.documentdb.uri}")
    private String uri;

    @Value("${azure.documentdb.key}")
    private String key;

    @Value("${azure.documentdb.database}")
    private String dbName;

    public DocumentClient documentClient() {
        return new DocumentClient(uri, key, ConnectionPolicy.GetDefault(), ConsistencyLevel.Session);
    }

    public String getDatabase() {
        return dbName;
    }
}
```



Select Operation

CreateDocument

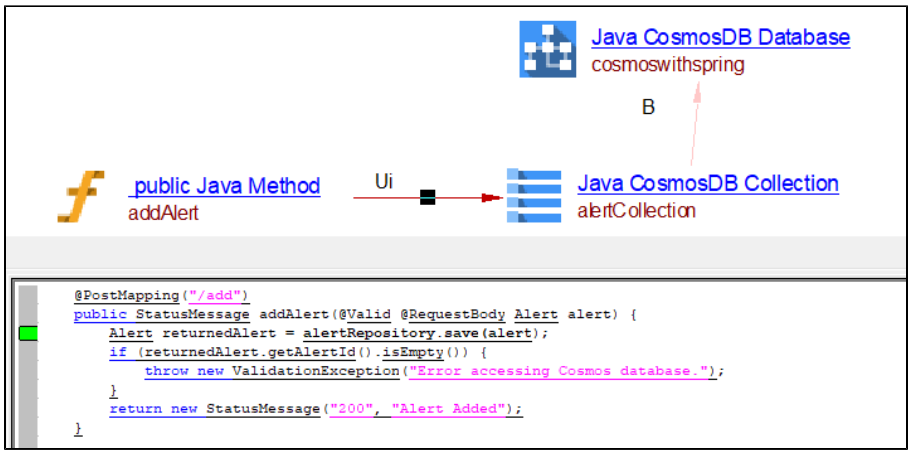
```
     @Autowired
    private AlertRepository alertRepository;

    @GetMapping("/all")
    public Iterable getAllAlerts() {
        if (StreamSupport.stream(alertRepository.findAll().spliterator(), false).count() > 0) {
            return alertRepository.findAll();
        } else {
            throw new ValidationException("No records found.");
        }
    }

 @GetMapping("/count")
    public long countAlert() {
        if (StreamSupport.stream(alertRepository.findAll().spliterator(), false).count() > 0) {
            return alertRepository.count();
        } else {
            throw new ValidationException("No records found.");
        }
    }
```



Insert Operation

QueryDocuments

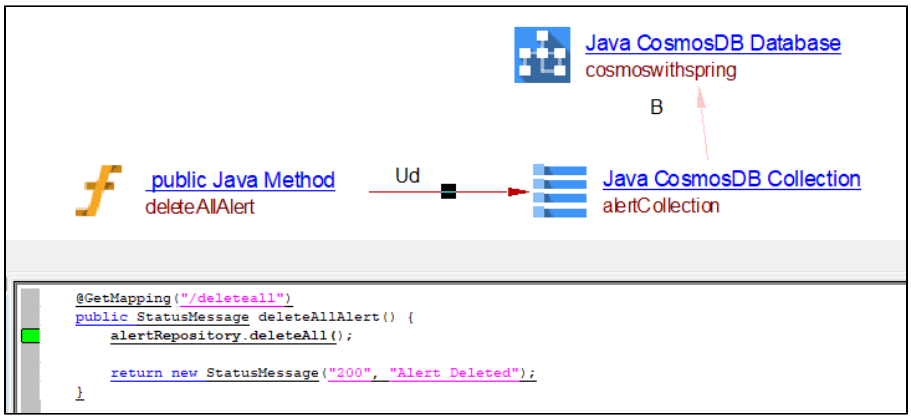
```
@PostMapping("/add")
   public StatusMessage addAlert(@Valid @RequestBody Alert alert) {
       Alert returnedAlert = alertRepository.save(alert);
       if (returnedAlert.getAlertId().isEmpty()) {
           throw new ValidationException("Error accessing Cosmos database.");
       }
       return new StatusMessage("200", "Alert Added");
   }
```

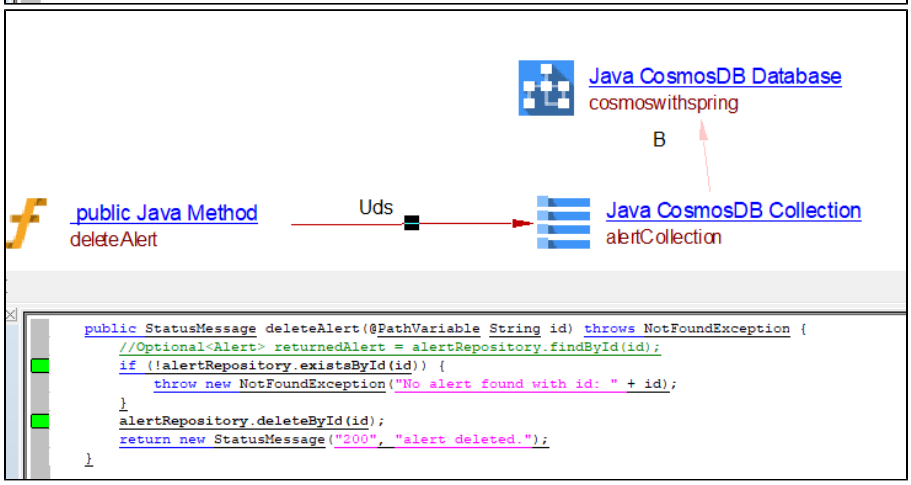


Delete Operation

DeleteDocument

```
@GetMapping("/deleteall")
    public StatusMessage deleteAllAlert() {
        alertRepository.deleteAll();

        return new StatusMessage("200", "Alert Deleted");
    }

 @DeleteMapping("/delete/{id}")
    public StatusMessage deleteAlert(@PathVariable String id) throws NotFoundException {
            //Optional<Alert> returnedAlert = alertRepository.findById(id);
        if (!alertRepository.existsById(id)) {
            throw new NotFoundException("No alert found with id: " + id);
        }
        alertRepository.deleteById(id);
        return new StatusMessage("200", "alert deleted.");
    }
```





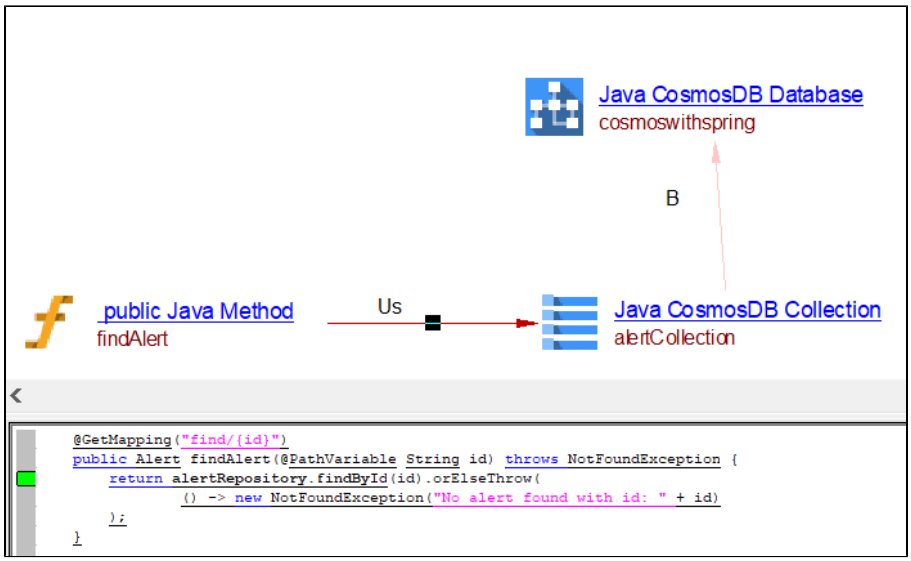
Query Method Support

Repo with query method

```
@Repository
public interface AlertRepository extends CosmosRepository<Alert, String> {
// repo for alert
        List<Alert> findById(String priority);
}
```

Controller

```
@GetMapping("find/{id}")
    public Alert findAlert(@PathVariable String id) throws NotFoundException {
        return alertRepository.findById(id).orElseThrow(
                () -> new NotFoundException("No alert found with id: " + id)
        );
}
```

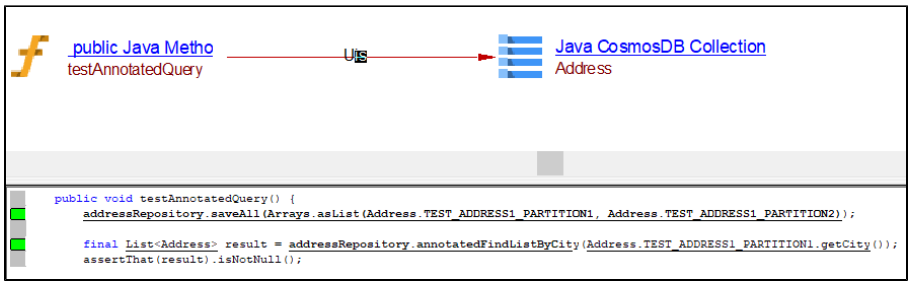


Query Methods with @Query annotation

Repository with @query methods

```
@Repository
public interface AddressRepository extends CosmosRepository<Address, String> {
    void deleteByPostalCodeAndCity(String postalCode, String city);

    void deleteByCity(String city);

    Iterable<Address> findByPostalCodeAndCity(String postalCode, String city);

    Iterable<Address> findByCity(String city);

    Iterable<Address> findByCityIn(List<String> cities);

    Iterable<Address> findByPostalCode(String postalCode);

    Iterable<Address> findByPostalCodeInAndCity(List<String> postalCodes, String city);

    Iterable<Address> findByStreetOrCity(String street, String city);

    @Query("select * from a where a.city = @city")
    List<Address> annotatedFindListByCity(@Param("city") String city);

    @Query("select * from a where a.city = @city")
    Page<Address> annotatedFindByCity(@Param("city") String city, Pageable pageable);
}
```

```
public void testAnnotatedQuery() {
        addressRepository.saveAll(Arrays.asList(Address.TEST_ADDRESS1_PARTITION1, Address.
TEST_ADDRESS1_PARTITION2));

        final List<Address> result = addressRepository.annotatedFindListByCity(Address.
TEST_ADDRESS1_PARTITION1.getCity());
        assertThat(result).isNotNull();
        assertThat(result.size()).isEqualTo(1);
        assertThat(result.get(0)).isEqualTo(Address.TEST_ADDRESS1_PARTITION1);
    }
```



Asynchronous API

Async Client creation

```
import com.microsoft.azure.spring.data.cosmosdb.repository.ReactiveCosmosRepository;

import info.hayslip.AlertHoarder.models.Alert;
import org.springframework.stereotype.Repository;

@Repository
public interface AlertReactRepository extends ReactiveCosmosRepository<Alert, String> {
// repo for alert
}
```

Evolutions

- Query methods are supported
- Query methods with @Query annotation are supported
- Better resolution for the collection names used in the crud operation
- Support for crud operations and query methods performed using class member such as "this."
- Support for CosmosOperations and CosmosTemplate.

Limitations

- Database/Collection is created as unknown, if the name is not retrieved from the properties file or if the name could not be resolved.