Introduction to DataSets

Module Objectives

In this module, you will learn:

- The purpose of DataSets
- The different types of DataSets
- Using DataSets in Applications

DataSets Store and Retrieve Data

- DataSets are the means of reading and writing data to and from the Reactor's storage capabilities
- DataSets provide:
    - Higher-level abstractions
    - A way to avoid manipulating data with low-level APIs
    - Generic, reusable Java implementations of common data patterns
- A DataSet represents both the API and the actual data itself
    - It is a named collection of data with associated metadata
    - It is manipulated through a DataSet class
    - A Java class that extends the abstract DataSet class with its own, custom methods
    - Implementation of a DataSet typically relies on one or more underlying (embedded) DataSets

Types of DataSets

Three categories of DataSets: *core*, *system*, and *custom* DataSets:

- The **core** DataSet of the Reactor is a Table
  - Its implementation is hidden from developers
  - It may use private DataSet interfaces that are not available to you
- A **system** DataSet is bundled with the Reactor
  - It is built around one or more underlying core or system DataSets to implement a specific data pattern
- A **custom** DataSet is implemented by you and can have arbitrary code and methods
  - It is typically built around one or more Tables (or other DataSets) to implement a specific data pattern
  - A custom DataSet can only manipulate data through its underlying DataSets

---

Core DataSet of Continuuity Reactor is a Table

- Unlike relational database systems, these tables are not organized into rows with a fixed schema
- They are optimized for efficient storage of:
    - Semi-structured data;
    - Data with unknown or variable schema; or
    - Sparse data
- Other DataSets are built on top of Tables

System and Custom DataSets

- A DataSet can implement specific semantics around a Table, such as:
  - a Key/value Table
  - a Counter Table
- Can combine multiple DataSets to create complex data patterns
- Example: An indexed Table can be implemented by using one Table for the data to index and a second Table for the index itself
- A number of useful DataSets—**system DataSets**—are included with Reactor, including **key/value** tables, **indexed** tables and **time series** tables
- You can implement your own data patterns as **custom DataSets** on top of Tables

---

System DataSets

- The `KeyValueTable` implements a key/value store as a Table with a single column
- The `IndexedTable` implements a Table with a secondary key using two embedded Tables, one for the data and one for the secondary index
- The `TimeseriesTable` uses a Table to store keyed data over time and allows querying that data over ranges of time

---

Using DataSets in Applications

To use a DataSet, you must declare it in the Application specification

To specify that your Application uses a `KeyValueTable` DataSet named *myCounters*, write:

```
public ApplicationSpecification configure() {
  return ApplicationSpecification.Builder.with()
    ...
    .withDataSets().add(new KeyValueTable("myCounters"))
    ...
```

Using DataSets in a Flowlet or a Procedure

To use a DataSet in a Flowlet or a Procedure, instruct the runtime system to inject an instance of the DataSet with the `@UseDataSet` annotation:

```
Class MyFowlet extends AbstractFlowlet {
  @UseDataSet("myCounters")
  private KeyValueTable counters;
  ...
  void process(String key) {
    counters.increment(key.getBytes());
  }
```

The runtime system:

- Reads the DataSet specification for the key/value table *myCounters* from the metadata store
- Injects a functional instance of the DataSet class into the Application

---

Module Summary

You should now be able to:

- Describe the three different types of DataSets
- Understand the difference between DataSet types and when to use them
- Implement using existing DataSets in an application

Module Completed