

# Model Examples: Simple Client-Service Interactions

Greg Riccardi

Department of Computer Science, Florida State University

riccardi@cs.fsu.edu, <http://www.nesc.ac.uk/~greg>

1 Processing request datasets .....	1
2 Processing SQL select queries .....	2
3 SQL update and DDL statements .....	3
4 Bulk load .....	4
4 Processing errors and response documents.....	5
5 Sample scenarios for performImmediate with delivery to/from 3 <sup>rd</sup> party.....	5
5.1 Sql query request dataset.....	6
5.2 Response dataset:.....	6

This document demonstrates how the most recently proposed DAIS conceptual model can support simple query and update processing and third party data delivery. In addition, the document uses an XML document schema that expresses the capabilities of the conceptual model. In many ways the current conceptual model is very close to the GGF7 specification document.

According to the conceptual model, a dataset is a collection of elements, each of which is a triplet (name, type, value). To interact with a data service, a client will create a dataset document containing requests for services and submit it to a perform method of the service. The service will respond with a dataset document containing the responses to the requests. Thus datasets are the values that are moved around between clients and services. A data activity session is able to retain state in the form of named datasets.

A dataset in XML looks like

```
<dataset>
  <element>
    <name> ... </name>
    <type> ... </type>
    <value> ... </value>
  </element>
  ...
</dataset>
```

## 1 Processing request datasets

A request document is a dataset that includes one or more request elements. That is, elements whose type is derived from request. E.g. Request.Query.SqlQuery. Similarly, a response document is one that includes one or more response elements. In the language of DAIS, each request element represents an instance of an activity type. Activities in DAIS are the fundamental operations that can be performed on data services.

From a `Request.Update.SqlUpdate` element, the response is an instance of type `Response.Update.SqlUpdate`. The response element includes status information plus references to the dataset elements that were included in the response dataset as a result of satisfying the request.

A data resource (dr) supports method `performImmediate` which executes the request elements of a single request dataset. A data activity session (das) allows the retention of state within the service and supports method `performAsynchronous` in addition to `performImmediate`. Additional examples and explanations including service interactions and the use of data activity sessions can be found in the document “DAIS Data Service Interactions and the SkyQuery Portal” [<http://www.nesc.ac.uk/~greg/modelexample2.pdf>].

The example request datasets that follow each contain one or more request elements. The response to a multi-element request is a dataset that combines the response elements of the requests. No change in meaning results from combining requests.

## 2 Processing SQL select queries

The document of Figure 1 is a request dataset that includes a single request element, of type `Request.Query.SqlQuery`. The value tag of the element (lines 5-7) contains an SQL select query. This document can be sent to the `performImmediate` method of a data resource for processing.

```
1      <DataSet>
2        <element>
3          <name>req1</name>
4          <type> Request.Query.SqlQuery </type>
5          <value>
6            <SqlQuery>select * from customer</SqlQuery>
7          </value>
8        </element>
9      </DataSet>
```

Figure 1 Request dataset for SQL select statement

Figure 2 shows the response of an appropriate data resource to the request of Figure 1. The response document has two elements. The first element, in lines 2-10, is a direct response to the request element. It repeats the request details in line 6 and gives the name of the dataset element (line 6) that contains the table produced by the select statement.

```
1      <DataSet>
2        <element>
3          <name>req1</name>
4          <type>Response.Query.SqlQuery</type>
5          <value>
6            <request><SqlQuery>select * from customer</SqlQuery></request>
7            <Result name="req1.resp1"/>
8            <status>complete</status>
9          </value>
10       </element>
11       <element>
12         <name>req1.resp1</name><!-- contains the result table -->
13       <type>
```

```

14         <xs:schema>--table schema here in XML--</xs:schema>
15     </type>
16     <value><Table>...</Table></value>
17 </element>
18 </DataSet>

```

Figure 2 Part of the response dataset for the request of Figure 1

The second element of the response dataset (lines 11-17) contains the query result table. The type is given by including an XML schema for the table. Line 14 is a place holder for the schema, which is not included. Finally, line 16 is where the rows of the table, in their XML form, will be.

Several XML formats are appropriate for representing relational tables in the result document. The Sun Java WebRowSet format and the Microsoft ADO.Net DataSet format are useful candidates.

### 3 SQL update and DDL statements

Update statements and data definition language (DDL) statements do not return tables as results, but rather return either an update count (for update statements) or no result (for DDL statements). Figures 3 and 4 show the request and response documents for a delete request and a drop table request.

```

1     <DataSet>
2     <element>
3         <name>upd1</name>
4         <type> Request.Update.SqlUpdate </type>
5         <value>
6             <SqlUpdate>delete from Customer where lastName='Doe'
7             </SqlUpdate>
8         </value>
9     </element>
10    <element>
11        <name>ddl1</name>
12        <type> Request.Update.SqlUpdate </type>
13        <value>
14            <SqlUpdate>drop table Movie</SqlUpdate>
15        </value>
16    </element>
17 </DataSet>

```

Figure 3 An update request dataset:

```

1     <DataSet>
2     <element>
3         <name>upd1</name>
4         <type>Response.Update.SqlUpdate</type>
5         <value>
6             <request><SqlUpdate>delete from Customer where lastName="Doe"
7             </SqlUpdate></request>
8             <RowCount>2</RowCount>
9             <status>complete</status>
10        </value>
11    </element>
12 </DataSet>

```

```

13         <name>ddl1</name>
14         <type>Response.Update.SqlUpdate</type>
15         <value>
16             <request><SqlUpdate>drop table Customer</SqlUpdate></request>
17             <status>complete</status>
18         </value>
19     </element>
20 </DataSet>

```

Figure 4 Response dataset for request of Figure 3

## 4 Bulk load

A create table request, as shown in Figure 5, is an instance of the more general bulk load operation. Figure 5 includes a request to create a table from a dataset element. That is, to interpret the type of the element as the schema of the table and the `The CreateTable` tag in line 6 specifies the name of the new table as an attribute. The schema and values for the new table are contained in the dataset element named `table1`, shown in lines 9-15.

```

1     <DataSet>
2     <element>
3         <name>req1</name>
4         <type> Request.Update.SqlCreateTable </type>
5         <value>
6             <CreateTable name="movie"><Input element="table1"/></CreateTable>
7         </value>
8     </element>
9     <element><!-- example of this element given in appendix -->
10        <name>table1</name>
11        <type>
12            <xs:schema>--table schema here in XML--</xs:schema>
13        </type>
14        <value><Table>...</Table></value>
15    </element>
16 </DataSet>

```

Figure 5 SQL create table request dataset

```

1     <DataSet>
2     <element>
3         <name>req1</name>
4         <type>Response.Update.SqlCreateTable </type>
5         <value>
6             <request>
7                 <CreateTable name="customer2"><Input element="table1"/>
8                 </CreateTable>
9             </request>
10            <RowCount>15</RowCount>
11        </value>
12    </element>
13 </DataSet>

```

Figure 6 Response dataset for create table request of Figure 5

## 4 Processing errors and response documents

Each request element has specific meaning to the dr. If the dr is incapable of performing the request, the response dataset includes an exception tag. Figure 7 shows an example

```
1      <element>
2        <name>req1</name>
3        <type>Response.Query.SqlQuery</type>
4        <value>
5          <SqlQuery>select * from cxstomer</SqlQuery>
6          <exception><type>SqlQueryError</type>
7            <message>No table named cxstomer in database</message>
8          </exception>
9        </value>
10     </element>
```

Figure 7 Example of response document with exception

The processing of other requests within the same document may or may not be effected by the error. The DAIS standard will allow implementers to enforce various strategies for exception handling. Examples of possible interpretations follow.

A linear model of request processing would require that each subsequent request also have an error response. In this context, "subsequent" means in the order processed. An atomic transaction model would require that all updates in the request be cancelled. We probably want the response to include all of the information generated by the successful execution, together with a rollback tag.

## 5 Sample scenarios for performImmediate with delivery to/from 3<sup>rd</sup> party

[The following has yet to be explained]

The request document of Figure 8 specifies that the result of the query is to be delivered by ftp and not returned within the response document. The SqlQuery tag includes an output tag (lines 7-9) that identifies the element named delivery1 is to be used as the data delivery recipe.

```
1      <DataSet>
2        <element>
3          <name>req1</name>
4          <type> Request.Query.SqlQuery </type>
5          <value>
6            <SqlQuery>select * from customer
7              <output>
8                <DataDeliveryRecipe element="delivery1"/>
9              </output>
10           </SqlQuery>
11         </value>
12       </element>
13     <element>
14       <name>delivery1</name>
```

```

15         <type>DeliveryRecipe</type>
16         <value>
17             <recipe>
18                 <ftp>
19                     <server>ftp.nesc.ac.uk</server>
20                     <file>/pub/customer.xml</file>
21                     <usr>ftp</usr>
22                 </ftp>
23             </recipe>
24         </value>
25     </element>
26 </DataSet>

```

Figure 8 Sql query request dataset with 3<sup>rd</sup> party delivery

The delivery recipe (lines 13-25) specifies an ftp delivery. Lines 18-22 list the server name, file name and user id of the ftp transfer.

Figure 9 shows the response to the request. No data is returned to the client.

```

1     <DataSet>
2     <element>
3         <name>req1</name><!-- repeats the query and identifies the result -->
4         <type>Response.Query.SqlQuery</type>
5         <value>
6             <request>
7                 <SqlQuery>select * from customer
8                 <output>
9                     <DataDeliveryRecipe element="delivery1"/>
10                </output>
11            </SqlQuery>
12        </request>
13        <status>complete</status>
14    </value>
15 </element>
16 </DataSet>

```

Figure 9 Response dataset from the request of Figure 8

## Appendix: Example of table schema and values in XML

This example is derived from the XML format generated by the Microsoft ADO.Net DataSet class. The document could be embedded in the document of Figure 5 as the input to a bulkload operation.

```

1     <DataSet xmlns="http://tempuri.org/">
2     <element>
3         <name>table1</name>
4         <type>
5             <xs:schema id="WebServiceData" xmlns=""
6                 xmlns:xs="http://www.w3.org/2001/XMLSchema"
7                 xmlns:msdata="urn:schemasmicrosoftcom:xmlmsdata">
8                 <xs:element name="movie">
9                     <xs:complexType>
10                        <xs:sequence>
11                            <xs:element name="movieId" type="xs:int" minOccurs="0" />

```

```
12         <xs:element name="title" type="xs:string" minOccurs="0" />
13     </xs:sequence>
14 </xs:complexType>
15 </xs:element>
16 </xs:schema>
17 </type>
18 <value>
19     <movie>
20         <movieId>101</movieId>
21         <title>The Thirtynine Steps</title>
22     </movie>
23     <movie>
24         <movieId>123</movieId>
25         <title>Annie Hall</title>
26     </movie>
27     <movie>
28         <movieId>145</movieId>
29         <title>Lady and the Tramp</title>
30     </movie>
31 </value>
32 </element>
33 </DataSet>
```