Request/Response Specifications

Definition of the standard used in DAIL/PAL methodology

By: Capt. Gal Kahana

Table of Contents:

1.Introduction

This document defines an XML standard used by DAIL/PAL methodology for building web IT applications. The standard defines how a client object requests an action from a server object and how the response from the server will look like. While maintaining a clear structure, the standard is flexible enough to enable a definition of any request.

But first, a small introduction of what DAIL/PAL methodology is required.
The methodology was developed when the author was working on an IT application which was MS DNA based on the regular 3 tiers:

1. thin web client - via MS Explorer web browser
2. business logic objects - entity components responsible for translating request from the client to real data manipulation commands, and also for acquiring data – all this by maintaining the business rules. Each components symbolizes a single entity in the application
3. data storage - SQL based relational database

Normally there's also an intermediate tier between 2 and 3 that is named the DAL – data access layer, which is encapsulated the different actions required for data acquiring and manipulation over the specific data bases. This allowed for an easy replacement of databases and acquiring some ADO objects for those data bases, regardless of their database provider.

So who does what?
The web client responds to user inputs and sends requests to the business logic objects (via XMLHTTP or in the ASP server code). The business logic objects either call each other for applying the business rules. They are also responsible for translating the request data into SQL (or any other manipulation code) that will go to the data storage layer to actually perform data manipulation or query on the data.

All of the communication is normally done by XML. XML is chosen mainly because of XMLHTTP requirements, but also because XML is a simple method of handing parameters that vary both in type and form. This way one may call to the same object for acquiring a list of items, through different filter type.

Normally, when writing a business logic object, 2 questions are raised:

1. How will the translations from XML to SQL (or SQLs) are to be made?
2. How is permission data applied to the requests, meaning – how to check if the user is allowed to an action, or how to limit the user ability to perform the action only on the items that the user is allowed to.

In order to give an answer to those 2 questions, 2 components were built:
First a component that replaced the DAL layer in the way of communicating to the data base but also knew how to translate XML requests to SQL commands (or other type of data manipulation commands). The method of doing so was by having specific XSLs –

one for each action specified by the business logic objects – that translated the XMLs to a sequence of commands.
A requirement for a standard XML came right away because that component had to know how to build a response XML to those queries that will be standard and how to expect requests to be, so that encapsulation of the data source type would be complete and internal to the component and the XSLs. And so the Request/Response standard was born. The standard specifies in a general form how a request should look like, and how a response should, but is not (and this is important to say, because some people think it's an SQL substitution) an actual language that a data source knows how to work on. It is more like a format.
The second component to handle the permission data application on each request already relied heavily on the standard, in the method of limiting the queries of the user to what it was allowed to and also in the method of censoring data from a response.
The methodology of using both components and the XML standard is named the DAIL/PAL methodology. The document will define both the standard requirements and also some suggestions for standardizing some parameters (as they were used in the project originating the methodology). Assume that all definitions are required, unless they appear under a paragraph starting with the phrase *"Suggestion"*.

2.The Request

A standard request includes a single action request to the server. It includes all the parameters for the action as it is intended to be sent alone to the server.
A template for a request looks like this:

```
<Request id="dataItemID" action="actionID">
        <Data>
                <VariableNameA/>
                ….
        </Data>
        <Profile>
                <ProfileFieldA>
                        <Item>specificFieldAValue</Item>
                        …
                </ProfileFieldA>
        </Profile>
</Request>
```

The surrounding tag is named "Request" and it should always include 2 attributes:

1. id – the type id of the item on which the action is executed. Normally this implies for the business object name that accepts this request. An id may be for example "Person" (for a request on person/s) or a "HotelReservation" for hotel reservation request etc.
2. action – logical action id, for example : "Retrieve", for data retrieval or "Delete" for entities deletion, or "updateCoursePrice" etc.

Both attributes are used to identify the response for this request when multiple requests are being made. Other attributes may also be included.

2.1 Data tag

The "Data" tag in a request includes the parameters for the "action" on a single item of the type defined by "id".
For example a Data tag may include tags for defining fields required in a retrieve request for each retrieved entity. In a creation query it may include fields and values for the newly created entity.

Each tag in the Data section should match a single parameter for the action. Its content and attributes are free. For example, a single field name for an entity in a data retrieval query:

```
<Data>
        <Name/> <!-- the name of the retrieved item -->
        <Age/>    <!-- the age of the retrieved item -->
</Data>
```

In the example each tag defines a different field to be retrieved (name and age).

Finally the "Data" tag itself may include any attribute.

Suggestion:

In order to enable usage of fields which have a similar meaning, one may use names that are the same but differ in their suffix. For example in order to retrieve a computed field like the organization in a company in which a person is according to an organization type (to have the person team, division, corporation retrieved in a single field) one may use a tag with the name : OrganizationByTypeX , where the X may be replaced by any suffix.
In those cases, it is normally required to pass more then just the field name, but some parameters to it. This may be achieved by using a "Params" tag. Each tag in the "Params" tag is named "Param" and holds an attribute named "id" that identifies the parameter. The "Param" holds the value. If a single "Param" is required, one may drop the "id".
If we continue with the above example then a "Data" section in a query for the person team and division will look something like:

```
<Data>
        <OrganizationByType1>
                <Params><Param id="OrganizationType">team</Param></Params>
        </OrganizationByType1>
        <OrganizationByType2>
                <Params><Param id="OrganizationType">division</Param></Params>
        </OrganizationByType2>
</Data>
```

The first field in under the Data tag (the one named "OrganizationByType1") represents a field with a single parameter, which is the "OrganizationType" (as the "id" attribute states), the parameter value is "team" which implies that this field means: The team of the person. The second field, in which the "OrganizationType" parameter value is "division" means: the division of the person.

2.2 Profile tag

The "Profile" tag in a request describes the profile of the entities on which the action is being made. The profile tag includes property tags, each symbolizing a different property of the profiled entities.
In order to define values for each property tag one places "Item" tags. There can be one or more "Item" tags in each property tag - this is already determined by the specific property tag. The reason one may place more then a single Item is when the user wishes to specify that the profiled entities may have either on of the values defined in the items for this certain property.
The following "Profile" tag defines the entities that have an ID property of one of: 1, 2, 3 or 4 and are of "red" Color:

```
<Profile>
	<ID>
		<Item>1</Item>
		<Item>2</Item>
		<Item>3</Item>
		<Item>4</Item>
	</ID>
	<Color>
		<Item>red</Item>
	</Color>
</Profile>
```

A property tag may not have an "Item" in a case where this property is a yes/no property and its inclusion alone symbolizes that profiled items should just have this property as yes.

Important! Pay attention that by the definition of the standard the tags inside the "Profile" relate to each other by an "And" logical operator, meaning that the action should act upon the items that have all the properties as they are defined by the property tags.

Once this is settled, it should be easy to limit a given profile of a request by simply concatenating new profile properties to the profile string. This is more helped by the fact that in a "Profile" (unlike "Data") one may place more then one instance of the same tag. For example, the following is legal:

```
<Profile>
	<A><Item>1</Item><Item>2</Item><Item>3</Item></A>
```

```
        <B><Item>S</Item></B>
        <A><Item>2</Item></A>
</Profile>
```

This (in the implementation) should mean the same as:

```
<Profile>
        <B><Item>S</Item></B>
        <A><Item>2</Item></A>
</Profile>
```

The reason behind allowing placing more then one instance of the same property is to easily implement the "search within these results" capability without having to analyze too much the original profile.

Suggestion:
Some properties of the profile (tags under the "Profile" tag) should be allowed for more then a simple comparison of matching values. For example, one may wish to update a certain field of all items which price is more then 20$. To do so it is suggested that a "type" property will be placed either in the "Item" tag or in the property tag and which value is an operator symbol. Example:

```
<Profile>
        <Price type="after"><Item>20</Item></Price>
</Profile>
```

The "after" value in "type" property defines that the profile will hold all the entities of a price more then 20.
More examples for a type value are:
"after+" - more or equal
"before" - less then
"before+" - less then or equal
"recursive" - for hierarchical profiles, this will mean all items that have this property value or a value that is considered to be "below" that value in the hierarchy.

2.2 Profile and Data in different types of queries

Suggestion:
The "Profile" and "Data" tags will not always exist in a query. The following general cases describe what usually should happen:

A. A data retrieval request - "Data" tag exists always to signal what are the properties that are required to be returned for each entity of the profile. If the request only wishes to return a list of the items, with no property an empty "Data" tag is to be

placed. A "Profile" tag will exist if the query wishes to retrieve only a subset of the items, and not all of them. To signal an empty "Profile" one may either use an empty "Profile" tag, but it is preferred that it will simply be dropped.

B. An entity creation request - "Data" tag exists to define the initial properties of the newly created entity. "Profile" tag will normally not be defined, unless it is required because of a certain permission limitation.

C. An entity removal request - "Data" tag will not exist. A "Profile" will exist to define the profile of the items to be removed.

D. An entity property update request - both "Data" and "Profile" tags will exist, defining what properties to update, with what values, and of which entities.

More complex actions should use the above templates to determine what is to be done.

A certain logical relationship should exist between the tags of "Data" and the tags of "Profile". For example, if a hotel reservation has a property of "PaymentMethod" then in both cases when one wishes to retrieve that value (i.e. it will be a tag in the "Data" section) or to profile by this values (i.e. it will be a tag in the "Profile" section) the tag name will be the same. This will make the usage of the server accepting this request easier.

3.The Response

A standard response holds a single response from the server to a request of the client.
It shall be reminded, that this is a spec of how to respond to a request when using this standard, as in the form of the returning value. By no means do this imply on how this should be implemented by a server, it only defines how it should look like coming out. The method of returning such a response for the above given type of requests, depends totally on the will of the server programmer (or her manager…of course)
A general template to a response looks like the following:

<**Response** id="dataItemID" action="actionID">
    <**Item** id="itemID">
        <A>…</A>
    </**Item**>
    ….
</**Response**>

The surrounding tag is named "Response" and holds 2 attributes: "id" and "action". Those attributes match the attributes in the "Request" and should hold the same values as in the request that originated that response.
There are 2 kinds of "Response"s. The first is a response to a data retrieval query, and the second is a response to a data modification query. The next sections will explain both separately.

3.1 Data retrieval response

A data retrieval response is a response to a request that queries properties values for a certain profile of items/entities. The response holds a list of the items and in each item a list of tags, each for a different property. The property tags that exist are those that were requested. The order of the property tags for each item is the same as the order of the tags in the "Data" section of the request. An item will always have an "id" attribute in which a key differing it from other items will reside.
For example, given the following request:

```
<Request id="HotelReservation" action="Retrieve">
        <Data>
                <Price/>
                <PaymentMethod/>
                <NumberOfRooms/>
        </Data>
        <Profile>
                <Price type="after+"><Item>200</Item></Price>
        </Profile>
</Request>
```

The response will look something like:

```
<Response id="HotelReservation" action="Retrieve">
        <Item id="4">
                <Price>300</Price>
                <PaymentMethod>CreditCard</PaymentMethod>
                <NumberOfRooms>3</NumberOfRooms>
        </Item>
        <Item id="6">
                <Price>200</Price>
                <PaymentMethod>Cash</PaymentMethod>
                <NumberOfRooms>1</NumberOfRooms>
        </Item>
        …..
</Response>
```

Pay attention that the order of tags in each item in the response is the same as the order of tags in the Data section, that each item has an id (this time it's the hotel reservation code). Sometimes it will be required that a single property tag in the request should result in more then one property tag in the response. For example assume that a request is sent that asks for the phones of the employees of department B. the Request looks something like this:

```
<Request id="Employee" action="Retrieve">
        <Data>
                <Name/>
```

```
            <Phone/>
        </Data>
        <Profile>
            <Department><Item>B</Item></Department>
        </Profile>
</Request>
```

The matching response may look something like this:

```
<Response id="Employee" action="Retrieve">
        <Item id="21424">
            <Name>John Erwin</Name>
            <Phone type="home">5673421</Phone>
            <Phone type="mobile">23424</Phone>
        </Item>
        <Item id="57632">
            <Name>Alf Roth</Name>
            <Phone type="home">543221</Phone>
            <Phone type="home1">21465</Phone>
            <Phone type="mobile">123123</Phone>
        </Item>
</Response>
```

Notice the "Phone" tags. Although the request holds a single tag, the response returns many - and even of various number (since Alf has 2 home phones, and John has 1).

This kind of field is named: "Inflating Field". All tags of an inflating field must be adjacent to maintain the order property of the relationship between a request and a response.

Suggestion:
If one wishes to separate one sub field of an inflating field from another, a property may be used to differ (as seen in the example - notice the "type" attribute)
End Suggestion.

As told in the request section, a request may contain an empty "Data" tag. In that case the list of items will contain "Item" tags with only an "id" attribute - meaning no tags are allowed inside (although more attributes in the "Item" tag are allowed).
An empty retrieve response - for a request in which the profile describes no items - will contain only the surrounding "Response" tag, with no items inside.

3.2 Data modification response

A response to a data modification query holds a single "Item" tag. The "Item" includes an "OK" tag, in which a free string that should indicate to the user the status of the action

(fail, succeeded, number of modified entities, nothing, or whatever…). More tags are allowed to further indicate the status or to give more data.
A simple template to a data modification response looks like this:

```
<Response id="itemTypeID" action="actionID">
        <Item>
                <OK> some text indicating the status…. </OK>
                ..
                ... more tags indicating the status
        </Item>
</Response>
```

4. Summery

The document introduces an XML standard for requests and responses as used in the DAIL/PAL methodology. The format of the requests and responses is used by the components handling the data access and permissions, for passing data around and manipulating it. Apart from defining the basic - required form of the Request and Response XMLs, it also suggests how to define some behaviors that are required for the application usage.
It is important to note that also the requirements are more of suggestions, and that the DAIL and PAL components demand only a subset of the given rules (those subsets are described in the documents defining the components behavior). Even though, it is important to respect those requirements for increasing the standardization of this Request/Response format for future implementations to come.