XEP-0099: IQ Query Action Protocol

Iain Shigeoka
mailto:iain@jivesoftware.com
xmpp:smirk@jabber.com

2003-06-25
Version 0.1

Status	Type	Short Name
Deferred	Standards Track	Not yet assigned

Standardizes behavior of <iq/> <query/> actions for generic query behavior.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction

There is a need for consistent query behavior amongst XMPP <iq/> protocols. Currently each protocol invents it's own, slightly different behavior for conducting query behavior to create, read, update, and delete (CRUD) recipient node data. This document defines a generic query acton protocol to standardize behavior across <iq/> protocols. In addition, we hope this standard will make other protocols easier to understand and implement by using a common core protocol.

2 <query/> Action Protocol

2.1 Description

The existing XMPP core protocol defines four <iq/> types (get, set, result, and error). Unfortunately, these four types don't include a 'delete' type, and the 'set' type must do double duty for 'create' and 'update'. Many protocols can benefit from a clear separation of create and update paralleling other query languages such as SQL.
Protocols complying with the <query/> action protocol use <iq/> 'set' to initiate all request-response interactions. The particular action to be taken MUST be set as an "action" attribute in the <iq/> <query/> sub-element. The action attribute MUST have a value of 'create', 'read', 'update', or 'delete'. Responses use the standard <iq/> 'result' and 'error' types. For backward compatibility, an <iq/> 'get' query is treated as equivalent to an <iq/> 'set' query with action of 'read'. Action protocols may require all or just a subset of these actions depending on the desired outcome.
In addition to the action attribute an optional "strict" attribute may be set in the <iq/> <query/> sub-element. The only valid values for strict is "true" or "false" (case sensitive). The strict behavior of actions causes more errors to be returned which tends to make protocols more robust but also more complex. Action protocols MUST define the default value of the "strict" attribute in the context of that protocol. In addition, some protocols may not wish to allow changing the strict default, so action protocols MUST declare whether the strict behavior of the protocol may be set in the <iq/> <query/> sub-element.

2.2 Actions

create	Creates/inserts new data on the recipient node.
read	Retrieves data from the recipient node.
update	Updates existing data on the recipient node.
delete	Deletes existing data on the recipient node.

2.3 Elements

The root element is query which is in a namespace defining the protocol in use. The query element MUST have an attribute named 'action' with values given in the previous table.

2.4 Error Codes

The following error codes apply to all action codes.

Code	Text	Description
406	Not Acceptable	The IQ query contents are not properly formatted for the action protocol.
503	Service Unavailable	The IQ query is sent to a JID that cannot handle the query.

3 Create Action

3.1 Description

The create action inserts new data on the recipient node. If the strict attribute is 'true' the create action fails if colliding data already exists on the recipient node. If the strict attribute is false, the create action will insert new data on the recipient node overwriting existing colliding data if it exists (e.g. equivalent to update).

3.2 Error Codes

Code	Text	Description
409	Conflict	The strict attribute is set to 'true' and colliding data exists on the recipient node.

3.3 Examples

Creating new data on the server using iq:private, and strict actions when no existing data is on the server.

Listing 1: Client Stores New Private Data

```
SENDER:
<iq type="set" id="1001">
  <query xmlns="jabber:iq:private" action="create" strict="true">
    <exodus xmlns="exodus:prefs">
      <defaultnick>Hamlet</defaultnick>
    </exodus>
  </query>
</iq>

RECIPIENT:
<iq
    type="result"
    from="hamlet@shakespeare.lit/denmark"
    to="hamlet@shakespeare.lit/denmark"
    id="1001"/>
```

With strict actions enabled, conflict data will cause the create action to fail when existing data is on the recipient node. Here we show iq:private, and strict actions with existing data on the server.

Listing 2: Client Stores New Private Data but Conflicts

```
SENDER:
<iq type="set" id="1002">
  <query xmlns="jabber:iq:private" action="create" strict="true">
    <exodus xmlns="exodus:prefs">
      <defaultnick>Hamlet</defaultnick>
    </exodus>
  </query>
</iq>

RECIPIENT:
<iq
    type="error"
    from="hamlet@shakespeare.lit/denmark"
    to="hamlet@shakespeare.lit/denmark"
    id="1002">
    <error code="409">Conflict</error>
    <exodus xmlns="exodus:prefs">
      <defaultnick>Hamlet</defaultnick>
    </exodus>
  </query>
</iq>
```

4 Read Action

4.1 Description

The read action retrieves data on the recipient node. If the strict attribute is 'true' the read action fails if no appropriate data exists on the recipient node. If the strict attribute is false, the read action will always return with a 'result', sending an empty result in place of a 'not found' error.

4.2 Error Codes

Code	Text	Description
404	Not found	The strict attribute is set to 'true' and no matching data exists on the recipient node.

4.3 Examples

Reading data on the server using iq:private, and strict actions when data is on the server.

Listing 3: Client Reads Private Data

```
SENDER:
<iq type="set" id="1001">
  <query xmlns="jabber:iq:private" action="read" strict="true">
    <exodus xmlns="exodus:prefs"/>
      <defaultnick>Hamlet</defaultnick>
    </exodus>
  </query>
</iq>

RECIPIENT:
<iq
    type="result"
    from="hamlet@shakespeare.lit/denmark"
    to="hamlet@shakespeare.lit/denmark"
    id="1001">
  <query xmlns="jabber:iq:private" action="read" strict="true">
    <exodus xmlns="exodus:prefs"/>
      <defaultnick>Hamlet</defaultnick>
    </exodus>
  </query>
</iq>
```

With strict actions enabled, the absence of matching data will cause the read action to fail. Here we show iq:private, and strict actions with no matching data on the server.

Listing 4: Client Reads Private Data but Not Found (strict)

```
SENDER:
<iq type="set" id="1002">
  <query xmlns="jabber:iq:private" action="read" strict="true">
    <data xmlns="imaginary"/>
  </query>
</iq>

RECIPIENT:
<iq
    type="error"
    from="hamlet@shakespeare.lit/denmark"
    to="hamlet@shakespeare.lit/denmark"
    id="1002">
    <error code="404">Not Found</error>
    <data xmlns="imaginary"/>
  </query>
</iq>
```

With strict actions disabled, the absence of matching data will cause the read action to return an 'empty' result. Here we show iq:private, and strict actions disabled with no matching data on the server.

Listing 5: Client Reads Private Data but Not Found (not strict)

```
SENDER:
<iq type="set" id="1003">
  <query xmlns="jabber:iq:private" action="read" strict="false">
    <data xmlns="imaginary"/>
  </query>
</iq>

RECIPIENT:
<iq
    type="result"
    from="hamlet@shakespeare.lit/denmark"
    to="hamlet@shakespeare.lit/denmark"
    id="1003">
    <data xmlns="imaginary"/>
  </query>
</iq>
```

5 Update Action

5.1 Description

The update action edits existing data on the recipient node. If the strict attribute is 'true' the update action fails if matching data does not already exists on the recipient node. If the strict attribute is false, the update action will edit existing data, inserting the data on the recipient node if necessary.

5.2 Error Codes

Code	Text	Description
404	Not Found	The strict attribute is set to 'true' and matching data does NOT already exist on the recipient node.

5.3 Examples

Updating existing new data on the server using iq:private, and strict actions when existing data is on the server.

Listing 6: Client Updates Existing Private Data

```
SENDER:
<iq type="set" id="1001">
  <query xmlns="jabber:iq:private" action="update" strict="true">
    <exodus xmlns="exodus:prefs">
      <defaultnick>Hamlet</defaultnick>
    </exodus>
  </query>
</iq>

RECIPIENT:
<iq
    type="result"
    from="hamlet@shakespeare.lit/denmark"
    to="hamlet@shakespeare.lit/denmark"
    id="1001"/>
```

With strict actions enabled, the absence of existing data will cause the update action to fail. Here we show iq:private, and strict actions with no existing data on the server.

Listing 7: Client Updates Private Data but None Found

```
SENDER:
<iq type="set" id="1002">
  <query xmlns="jabber:iq:private" action="update" strict="true">
    <exodus xmlns="exodus:prefs">
      <defaultnick>Hamlet</defaultnick>
    </exodus>
  </query>
</iq>

RECIPIENT:
<iq
    type="error"
    from="hamlet@shakespeare.lit/denmark"
    to="hamlet@shakespeare.lit/denmark"
    id="1002">
    <error code="404">Not Found</error>
    <exodus xmlns="exodus:prefs">
      <defaultnick>Hamlet</defaultnick>
    </exodus>
  </query>
</iq>
```

6 Delete Action

6.1 Description

The delete action deletes existing data on the recipient node. If the strict attribute is 'true' the delete action fails if matching data does not already exists on the recipient node. If the strict attribute is false, the delete action will delete any existing data on the recipient node (if any) and return successful..

6.2 Error Codes

Code	Text	Description
404	Not Found	The strict attribute is set to 'true' and matching data does NOT already exist on the recipient node.

6.3 Examples

Deleting existing new data on the server using iq:private, and strict actions when existing data is on the server.

Listing 8: Client Deletes Existing Private Data

```
SENDER:
<iq type="set" id="1001">
  <query xmlns="jabber:iq:private" action="delete" strict="true">
    <exodus xmlns="exodus:prefs"/>
  </query>
</iq>

RECIPIENT:
<iq
    type="result"
    from="hamlet@shakespeare.lit/denmark"
    to="hamlet@shakespeare.lit/denmark"
    id="1001"/>
```

With strict actions enabled, the absence of existing data will cause the delete action to fail. Here we show iq:private, and strict actions with no existing data on the server.

Listing 9: Client Deletes Private Data but None Found (strict)

```
SENDER:
<iq type="set" id="1002">
  <query xmlns="jabber:iq:private" action="delete" strict="true">
    <data xmlns="imaginary"/>
  </query>
</iq>

RECIPIENT:
<iq
    type="error"
    from="hamlet@shakespeare.lit/denmark"
    to="hamlet@shakespeare.lit/denmark"
    id="1002">
    <error code="404">Not Found</error>
    <data xmlns="imaginary"/>
  </query>
</iq>
```

With strict actions disabled, the absence of existing data will not cause the delete action to fail. Here we show iq:private, and strict actions with no existing data on the server.

Listing 10: Client Deletes Private Data but None Found (not strict)

```
SENDER:
<iq type="set" id="1003">
  <query xmlns="jabber:iq:private" action="delete" strict="false">
    <data xmlns="imaginary"/>
  </query>
</iq>

RECIPIENT:
<iq
    type="result"
    from="hamlet@shakespeare.lit/denmark"
    to="hamlet@shakespeare.lit/denmark"
    id="1003"/>
```

7 Defining an Action Protocol

7.1 Description

In order to define an action protocol that uses the <query/> behavior defined in this document, you must specify the following:

- The actions (create, read, update, delete) supported in the action protocol.
- The matching semantics for determing if data exists/collides.
- The default "strict" attribute ('true' or 'false'). This may be defined for each action supported or for all actions supported.
- Whether the "strict" attribute may be set by the user. If the attribute may not be set, the strict attribute will always hold the default value.