XEP-0004: Data Forms

Ryan Eatmon
mailto:reatmon@jabber.org
xmpp:reatmon@jabber.org

Joe Hildebrand
mailto:jhildebr@cisco.com
xmpp:hildjj@jabber.org

Jeremie Miller
mailto:jer@jabber.org
xmpp:jer@jabber.org

Thomas Muldowney
mailto:temas@jabber.org
xmpp:temas@jabber.org

Peter Saint-Andre
mailto:peter@andyet.net
xmpp:stpeter@stpeter.im
https://stpeter.im/

2007-08-13
Version 2.9

Status	Type	Short Name
Final	Standards Track	x-data

This specification defines an XMPP protocol extension for data forms that can be used in workflows such as service configuration as well as for application-specific data description and reporting. The protocol includes lightweight semantics for forms processing (such as request, response, submit, and cancel), defines several common field types (boolean, list options with single or multiple choice, text with single line or multiple lines, single or multiple JabberIDs, hidden fields, etc.), provides extensibility for future data types, and can be embedded in a wide range of applications. The protocol is not intended to provide complete forms-processing functionality as is provided in the W3C XForms technology, but instead provides a basic subset of such functionality for use by XMPP entities.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction

Several existing Jabber/XMPP protocols involve the exchange of structured data between users and applications for common tasks such as registration (In-Band Registration (XEP-0077) [1]) and searching (Jabber Search (XEP-0055) [2]). Unfortunately, these early protocols were ”hard coded” and thus place significant restrictions on the range of information that can be exchanged. Furthermore, other protocols (e.g., Multi-User Chat (XEP-0045) [3]) may need to exchange data for purposes such as configuration, but the configuration options may differ depending on the specific implementation or deployment. Finally, developers may want to extend other protocols (e.g., Service Discovery (XEP-0030) [4]) in a flexible manner in order to provide information that is not defined in the base protocol. In all of these cases, it would be helpful to use a generic data description format that can be used for dynamic forms generation and data ”modelling” in a variety of circumstances.

An example may be helpful. Let us imagine that when a user creates a multi-user chatroom on a text conferencing service, the service allows the user to configure the room in various ways. While most implementations will probably provide a somewhat common set of configurable features (discussion logging, maximum number of room occupants, etc.), there will be some divergence: perhaps one implementation will enable archiving of the room log in a variety of file types (XML, HTML, PDF, etc.) and for a variety of time periods (hourly, daily, weekly, etc.), whereas another implementation may present a boolean on/off choice of logging in only one format (e.g., daily logs saved in HTML). Obviously, the first implementation will have more configuration options than the second implementation. Rather than ”hard-coding” every option via distinct XML elements (e.g., <room_logging_period/>), a better design would involve a more flexible format.

The ’jabber:x:data’ protocol described herein defines such a flexible format for use by Jabber/XMPP entities, steering a middle course between the simplicity of ”name-value” pairs and the complexity of XForms 1.0 [5] (on which development had just begun when this protocol was designed). In many ways, ’jabber:x:data’ is similar to the Forms Module of XHTML 1.0 [6]; however, it provides several Jabber-specific data types, enables applications to require data fields, integrates more naturally into the ”workflow” semantics of IQ stanzas, and can be included as an extension of existing Jabber/XMPP protocols in ways that the XHTML Forms Module could not when this protocol was developed (especially because Modularization of XHTML [7] did not exist at that time).

[1] XEP-0077: In-Band Registration <http://xmpp.org/extensions/xep-0077.html>.
[2] XEP-0055: Jabber Search <http://xmpp.org/extensions/xep-0055.html>.
[3] XEP-0045: Multi-User Chat <http://xmpp.org/extensions/xep-0045.html>.
[4] XEP-0030: Service Discovery <http://xmpp.org/extensions/xep-0030.html>.
[5] XForms 1.0 <http://www.w3.org/TR/xforms>.
[6] XHTML 1.0 <http://www.w3.org/TR/xhtml1>.
[7] Modularization of XHTML <http://www.w3.org/TR/2004/WD-xhtml-modularization-20040218/>.

2 Requirements

This document addresses the following requirements:

1. **Data Gathering** -- the protocol should enable a form-processing entity (commonly a server, service, or bot) to gather data from a form-submitting entity (commonly a client controlled by a human user); this should be done via distinct data fields (e.g., items in a questionnaire or configuration form), each of which can be a different data "type" and enable free-form input or a choice between multiple options (as is familiar from HTML forms).
2. **Data Reporting** -- the protocol should enable a form-processing entity to report data (e.g., search results) to a form-submitting entity, again via distinct data fields.
3. **Portability** -- the protocol should as much as possible define generic data formats and basic datatypes only; hints may be provided regarding the user interface, but they should be hints only and not hard-and-fast requirements.
4. **Simplicity** -- the protocol should be simple for clients to implement, and most of the complexity (e.g., data validation and processing) should be the responsibility of servers and components rather than clients.
5. **Flexibility** -- the protocol should be flexible and extensible rather than "hard-coded".
6. **Compatibility** -- the protocol should define an extension to existing Jabber/XMPP protocols and not break existing implementations unless absolutely necessary.

3 Protocol

The base syntax for the 'jabber:x:data' namespace is as follows (a formal description can be found in the XML Schema section below):

```
<x xmlns='jabber:x:data'
   type='{form-type}'>
  <title/>
  <instructions/>
  <field var='field-name'
         type='{field-type}'
         label='description'>
    <desc/>
    <required/>
    <value>field-value</value>
    <option label='option-label'><value>option-value</value></option>
    <option label='option-label'><value>option-value</value></option>
  </field>
</x>
```

The <x/> element qualified by the 'jabber:x:data' namespace SHOULD be included either directly as a first-level child of a <message/> stanza or as a second-level child of an <iq/> stanza (where the first-level child is an element qualified by a "wrapper" namespace); see also the restrictions enumerated below.
The OPTIONAL <title/> and <instructions/> elements enable the form-processing entity to label the form as a whole and specify natural-language instructions to be followed by the form-submitting entity. The XML character data for these elements SHOULD NOT contain newlines (the \n and \r characters), and any handling of newlines (e.g., presentation in a user interface) is unspecified herein; however, multiple instances of the <instructions/> element MAY be included.

3.1 Form Types

The data gathered or provided in a 'jabber:x:data' form can be situated in a number of different contexts. Examples include an empty form that needs to be filled out, a completed form, the results of a submission, a search result, or simply a set of data that is encapsulated using the 'jabber:x:data' namespace. The full context for the data is provided by three things:

1. the "wrapper" protocol (i.e., the namespace whose root element is the direct child of the <iq/> stanza and the parent of the <x/> element qualified by the 'jabber:x:data' namespace)
2. the place of the form within a transaction (e.g., an IQ "set" or "result") or structured conversation (e.g., a message <thread/>)
3. the 'type' attribute on the form's root <x/> element

The first two pieces of contextual information are provided by other protocols, whereas the form types are described in the following table.

Type	Description
form	The form-processing entity is asking the form-submitting entity to complete a form.
submit	The form-submitting entity is submitting data to the form-processing entity. The submission MAY include fields that were not provided in the empty form, but the form-processing entity MUST ignore any fields that it does not understand.
cancel	The form-submitting entity has cancelled submission of data to the form-processing entity.
result	The form-processing entity is returning data (e.g., search results) to the form-submitting entity, or the data is a generic data set.

In order to maintain the context of the data as captured in the form type, the following rules MUST be observed:

- For <iq/> stanzas, the root element qualified by the "wrapper" namespace in a form of type "form" or "submit" MUST be returned in a form of type "result". The <x/> element qualified by the 'jabber:x:data' namespace MUST be a child of the "wrapper" namespace's root element. As defined in XMPP Core [8], the 'id' attribute MUST be copied in the IQ result. For data forms of type "form" or "result", the <iq/> stanza SHOULD be of type "result". For data forms of type "submit" or "cancel", the <iq/> stanza SHOULD be of type "set".

- For <message/> stanzas, the <thread/> SHOULD be copied in the reply if provided. The <x/> element qualified by the 'jabber:x:data' namespace MUST be a child of the <message/> stanza.

3.2 The Field Element

A data form of type "form", "submit", or "result" SHOULD contain at least one <field/> element; a data form of type "cancel" SHOULD NOT contain any <field/> elements.
The <field/> element MAY contain any of the following child elements:

<desc/> The XML character data of this element provides a natural-language description of the field, intended for presentation in a user-agent (e.g., as a "tool-tip", help button, or explanatory text provided near the field). The <desc/> element SHOULD NOT contain newlines (the \n and \r characters), since layout is the responsibility of a user agent, and any handling of newlines (e.g., presentation in a user interface) is unspecified herein. (Note: To provide a description of a field, it is RECOMMENDED to use a <desc/> element rather than a separate <field/> element of type "fixed".)

<required/> This element, which MUST be empty, flags the field as required in order for the form to be considered valid.

<value/> The XML character data of this element defines the default value for the field (according to the form-processing entity) in a data form of type "form", the data provided by a form-submitting entity in a data form of type "submit", or a data result in a data form of type "result". In data forms of type "form", if the form-processing entity provides a default value via the <value/> element, then the form-submitting entity SHOULD NOT attempt to enforce a different default value (although it MAY do so to respect user preferences or anticipate expected user input). Fields of type list-multi, jid-multi, text-multi, and hidden MAY contain more than one <value/> element; all other field types MUST NOT contain more than one <value/> element.

[8] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.

<option/> One of the options in a field of type ”list-single” or ”list-multi”. The XML character of the <value/> child defines the option value, and the ’label’ attribute defines a human-readable name for the option. The <option/> element MUST contain one and only one <value/> child. If the field is not of type ”list-single” or ”list-multi”, it MUST NOT contain an <option/> element.

If the <field/> element type is anything other than ”fixed” (see below), it MUST possess a ’var’ attribute that uniquely identifies the field in the context of the form (if it is ”fixed”, it MAY possess a ’var’ attribute). The <field/> element MAY possess a ’label’ attribute that defines a human-readable name for the field. For data forms of type ”form”, each <field/> element SHOULD possess a ’type’ attribute that defines the data ”type” of the field data (if no ’type’ is specified, the default is ”text-single”); fields provided in the context of other forms types MAY possess a ’type’ attribute as well. For data forms of type ”submit”, inclusion of the ’type’ attribute is OPTIONAL, since the form-processing entity is assumed to understand the data types associated with forms that it processes.
If fields are presented in a user interface (e.g., as items in a questionnaire or form result), the order of the field elements in the XML SHOULD determine the order of items presented to the user.

3.3 Field Types

The following field types represent data ”types” that are commonly exchanged between Jabber/XMPP entities. These field types are not intended to be as comprehensive as the datatypes defined in, for example, XML Schema Part 2 [9], nor do they define user interface elements.

Type	Description
boolean	The field enables an entity to gather or provide an either-or choice between two options. The default value is ”false”. In accordance with Section 3.2.2.1 of XML Schema Part 2: Datatypes, the allowable lexical representations for the xs:boolean datatype are the strings ”0” and ”false” for the concept ’false’ and the strings ”1” and ”true” for the concept ’true’; implementations MUST support both styles of lexical representation.
fixed	The field is intended for data description (e.g., human-readable text such as ”section” headers) rather than data gathering or provision. The <value/> child SHOULD NOT contain newlines (the \n and \r characters); instead an application SHOULD generate multiple fixed fields, each with one <value/> child.
hidden	The field is not shown to the form-submitting entity, but instead is returned with the form. The form-submitting entity SHOULD NOT modify the value of a hidden field, but MAY do so if such behavior is defined for the ”using protocol”.

[9] XML Schema Part 2: Datatypes <http://www.w3.org/TR/xmlschema-2/>.

Type	Description
jid-multi	The field enables an entity to gather or provide multiple Jabber IDs. Each provided JID SHOULD be unique (as determined by comparison that includes application of the Nodeprep, Nameprep, and Resourceprep profiles of Stringprep as specified in XMPP Core), and duplicate JIDs MUST be ignored. *
jid-single	The field enables an entity to gather or provide a single Jabber ID. *
list-multi	The field enables an entity to gather or provide one or more options from among many. A form-submitting entity chooses one or more items from among the options presented by the form-processing entity and MUST NOT insert new options. The form-submitting entity MUST NOT modify the order of items as received from the form-processing entity, since the order of items MAY be significant.**
list-single	The field enables an entity to gather or provide one option from among many. A form-submitting entity chooses one item from among the options presented by the form-processing entity and MUST NOT insert new options. **
text-multi	The field enables an entity to gather or provide multiple lines of text. ***
text-private	The field enables an entity to gather or provide a single line or word of text, which shall be obscured in an interface (e.g., with multiple instances of the asterisk character).
text-single	The field enables an entity to gather or provide a single line or word of text, which may be shown in an interface. This field type is the default and MUST be assumed if a form-submitting entity receives a field type it does not understand.

* Note: Data provided for fields of type ”jid-single” or ”jid-multi” MUST contain one or more valid Jabber IDs, where validity is determined by the addressing rules defined in XMPP Core (see the Data Validation section below).
* Note: The <option/> elements in list-multi and list-single fields MUST be unique, where uniqueness is determined by the value of the ’label’ attribute and the XML character data of the <value/> element (i.e., both must be unique).
** Note: Data provided for fields of type ”text-multi” SHOULD NOT contain any newlines (the \n and \r characters). Instead, the application SHOULD split the data into multiple strings (based on the newlines inserted by the platform), then specify each string as the XML character data of a distinct <value/> element. Similarly, an application that receives multiple <value/> elements for a field of type ”text-multi” SHOULD merge the XML character data of the value elements into one text block for presentation to a user, with each string separated by a newline character as appropriate for that platform.

3.4 Multiple Items in Form Results

In some contexts (e.g., the results of a search request), it may be necessary to communicate multiple items. Therefore, a data form of type ”result” MAY contain two child elements not

described in the basic syntax above:

1. One and only <reported/> element, which can be understood as a ”table header” describing the data to follow.
2. Zero or more <item/> elements, which can be understood as ”table cells” containing data (if any) that matches the request.

The syntax is as follows:

```
<x xmlns='jabber:x:data'
   type='result'>
  <reported>
    <field var='field-name' label='description' type='{field-type}'/>
  </reported>
  <item>
    <field var='field-name'>
      <value>field-value</value>
    </field>
  </item>
  <item>
    <field var='field-name'>
      <value>field-value</value>
    </field>
  </item>
  .
  .
  .
</x>
```

Each of these elements MUST contain one or more <field/> children. The <reported/> element defines the data format for the result items by specifying the fields to be expected for each item; for this reason, the <field/> elements SHOULD possess a 'type' attribute and 'label' attribute in addition to the 'var' attribute, and SHOULD NOT contain a <value/> element. Each <item/> element defines one item in the result set, and MUST contain each field specified in the <reported/> element (although the XML character data of the <value/> element MAY be null).

4 Data Validation

Data validation is the responsibility of the form-processing entity (commonly a server, service, or bot) rather than the form-submitting entity (commonly a client controlled by a human user). This helps to meet the requirement for keeping client implementations simple. If the form-processing entity determines that the data provided is not valid, it SHOULD return a ”Not Acceptable” error, optionally providing a textual explanation in the XMPP

<text/> element or an application-specific child element that identifies the problem (see Error Condition Mappings (XEP-0086) [10] for information about mappings and formats).

5 Examples

For the sake of the following examples, let us suppose that there exists a bot hosting service on the Jabber network, located at <botster.shakespeare.lit>. This service enables registered users to create and configure new bots, find and interact with existing bots, and so on. We will assume that these interactions occur using the Ad-Hoc Commands (XEP-0050) [11] protocol, which is used as a "wrapper" protocol for data forms qualified by the 'jabber:x:data' namespace. The examples in the sections that follow show most of the features of the data forms protocol described above.
Note: Additional examples can be found in the specifications for various "using protocols", such as XEP-0045: Multi-User Chat and XEP-0055: Jabber Search.

5.1 Configuration

The first step is for a user to create a new bot on the hosting service. We will assume that this is done by sending a "create" command to the desired bot:

Listing 1: User Requests Bot Creation

```
<iq from='romeo@montague.net/home'
    to='joogle@botster.shakespeare.lit'
    type='get'
    xml:lang='en'
    id='create1'>
  <command xmlns='http://jabber.org/protocol/commands'
           node='create'
           action='execute'/>
</iq>
```

The hosting service then returns a data form to the user:

Listing 2: Service Returns Bot Creation Form

```
<iq from='joogle@botster.shakespeare.lit'
    to='romeo@montague.net/home'
    type='result'
    xml:lang='en'
    id='create1'>
```

[10] XEP-0086: Error Condition Mappings <http://xmpp.org/extensions/xep-0086.html>.
[11] XEP-0050: Ad-Hoc Commands <http://xmpp.org/extensions/xep-0050.html>.

```
<command xmlns='http://jabber.org/protocol/commands'
         node='create'
         sessionid='create:20040408T0128Z'
         status='executing'>
  <x xmlns='jabber:x:data' type='form'>
    <title>Bot Configuration</title>
    <instructions>Fill out this form to configure your new bot!</
       instructions>
    <field type='hidden'
           var='FORM_TYPE'>
      <value>jabber:bot</value>
    </field>
    <field type='fixed'><value>Section 1: Bot Info</value></field>
    <field type='text-single'
           label='The␣name␣of␣your␣bot'
           var='botname'/>
    <field type='text-multi'
           label='Helpful␣description␣of␣your␣bot'
           var='description'/>
    <field type='boolean'
           label='Public␣bot?'
           var='public'>
      <required/>
    </field>
    <field type='text-private'
           label='Password␣for␣special␣access'
           var='password'/>
    <field type='fixed'><value>Section 2: Features</value></field>
    <field type='list-multi'
           label='What␣features␣will␣the␣bot␣support?'
           var='features'>
      <option label='Contests'><value>contests</value></option>
      <option label='News'><value>news</value></option>
      <option label='Polls'><value>polls</value></option>
      <option label='Reminders'><value>reminders</value></option>
      <option label='Search'><value>search</value></option>
      <value>news</value>
      <value>search</value>
    </field>
    <field type='fixed'><value>Section 3: Subscriber List</value></
       field>
    <field type='list-single'
           label='Maximum␣number␣of␣subscribers'
           var='maxsubs'>
      <value>20</value>
      <option label='10'><value>10</value></option>
      <option label='20'><value>20</value></option>
      <option label='30'><value>30</value></option>
      <option label='50'><value>50</value></option>
```

```
        <option label='100'><value>100</value></option>
        <option label='None'><value>none</value></option>
      </field>
      <field type='fixed'><value>Section 4: Invitations</value></field
         >
      <field type='jid-multi'
             label='People to invite'
             var='invitelist'>
        <desc>Tell all your friends about your new bot!</desc>
      </field>
    </x>
  </command>
</iq>
```

The user then submits the configuration form:

Listing 3: User Submits Bot Creation Form

```
<iq from='romeo@montague.net/home'
    to='joogle@botster.shakespeare.lit'
    type='set'
    xml:lang='en'
    id='create2'>
  <command xmlns='http://jabber.org/protocol/commands'
           node='create'
           sessionid='create:20040408T0128Z'>
    <x xmlns='jabber:x:data' type='submit'>
      <field type='hidden' var='FORM_TYPE'>
        <value>jabber:bot</value>
      </field>
      <field type='text-single' var='botname'>
        <value>The Jabber Google Bot</value>
      </field>
      <field type='text-multi' var='description'>
        <value>This bot enables you to send requests to</value>
        <value>Google and receive the search results right</value>
        <value>in your Jabber client. It&apos; really cool!</value>
        <value>It even supports Google News!</value>
      </field>
      <field type='boolean' var='public'>
        <value>0</value>
      </field>
      <field type='text-private' var='password'>
        <value>v3r0na</value>
      </field>
      <field type='list-multi' var='features'>
        <value>news</value>
        <value>search</value>
      </field>
```

```
      <field type='list-single' var='maxsubs'>
        <value>50</value>
      </field>
      <field type='jid-multi' var='invitelist'>
        <value>juliet@capulet.com</value>
        <value>benvolio@montague.net</value>
      </field>
    </x>
  </command>
</iq>
```

The service then returns the results to the user:

Listing 4: Service Returns Bot Creation Result

```
<iq from='joogle@botster.shakespeare.lit'
    to='romeo@montague.net/home'
    type='result'
    xml:lang='en'
    id='create2'>
  <command xmlns='http://jabber.org/protocol/commands'
           node='create'
           sessionid='create:20040408T0128Z'
           status='completed'>
    <x xmlns='jabber:x:data' type='result'>
      <field type='hidden' var='FORM_TYPE'>
        <value>jabber:bot</value>
      </field>
      <field type='text-single' var='botname'>
        <value>The Jabber Google Bot</value>
      </field>
      <field type='boolean' var='public'>
        <value>0</value>
      </field>
      <field type='text-private' var='password'>
        <value>v3r0na</value>
      </field>
      <field type='list-multi' var='features'>
        <value>news</value>
        <value>search</value>
      </field>
      <field type='list-single' var='maxsubs'>
        <value>50</value>
      </field>
      <field type='jid-multi' var='invitelist'>
        <value>juliet@capulet.com</value>
        <value>benvolio@montague.net</value>
      </field>
    </x>
```

```
  </command>
</iq>
```

5.2 Search

Now that the user has created this search bot, let us suppose that one of the friends he has invited decides to try it out by sending a search request:

Listing 5: User Requests Search Form

```
<iq from='juliet@capulet.com/chamber'
    to='joogle@botster.shakespeare.lit'
    type='get'
    xml:lang='en'
    id='search1'>
  <command xmlns='http://jabber.org/protocol/commands'
           node='search'
           action='execute'/>
</iq>
```

Listing 6: Service Returns Search Form

```
<iq from='joogle@botster.shakespeare.lit'
    to='juliet@capulet.com/chamber'
    type='result'
    xml:lang='en'
    id='search1'>
  <command xmlns='http://jabber.org/protocol/commands'
           node='search'
           status='executing'>
    <x xmlns='jabber:x:data' type='form'>
      <title>Joogle Search</title>
      <instructions>Fill out this form to search for information!</
          instructions>
      <field type='text-single'
             var='search_request'>
        <required/>
      </field>
    </x>
  </command>
</iq>
```

Listing 7: User Submits Search Form

```
<iq from='juliet@capulet.com/chamber'
    to='joogle@botster.shakespeare.lit'
    type='get'
    xml:lang='en'
```

```
    id='search2'>
  <command xmlns='http://jabber.org/protocol/commands'
           node='search'>
    <x xmlns='jabber:x:data' type='submit'>
      <field type='text-single' var='search_request'>
        <value>verona</value>
      </field>
    </x>
  </command>
</iq>
```

Listing 8: Service Returns Search Results

```
<iq from='joogle@botster.shakespeare.lit'
    to='juliet@capulet.com/chamber'
    type='result'
    xml:lang='en'
    id='search2'>
  <command xmlns='http://jabber.org/protocol/commands'
           node='search'
           status='completed'>
    <x xmlns='jabber:x:data' type='result'>
      <title>Joogle Search: verona</title>
      <reported>
        <field var='name'/>
        <field var='url'/>
      </reported>
      <item>
        <field var='name'>
          <value>Comune di Verona - Benvenuti nel sito ufficiale</
             value>
        </field>
        <field var='url'>
          <value>http://www.comune.verona.it/</value>
        </field>
      </item>
      <item>
        <field var='name'>
          <value>benvenuto!</value>
        </field>
        <field var='url'>
          <value>http://www.hellasverona.it/</value>
        </field>
      </item>
      <item>
        <field var='name'>
          <value>Universita degli Studi di Verona - Home Page</value>
        </field>
        <field var='url'>
          <value>http://www.univr.it/</value>
```

```
        </field>
      </item>
      <item>
        <field var='name'>
          <value>Aeroporti del Garda</value>
        </field>
        <field var='url'>
          <value>http://www.aeroportoverona.it/</value>
        </field>
      </item>
      <item>
        <field var='name'>
          <value>Veronafiere - fiera di Verona</value>
        </field>
        <field var='url'>
          <value>http://www.veronafiere.it/</value>
        </field>
      </item>
    </x>
  </command>
</iq>
```

6 Service Discovery

If an entity supports inclusion of the <x/> element qualified by the 'jabber:x:data' namespace as a direct child of the <message/> stanza type, it MUST report support by including a service discovery feature of "jabber:x:data" (see Protocol Namespaces regarding issuance of one or more permanent namespaces) in response to a Service Discovery information request:

Listing 9: Service Discovery information request

```
<iq type='get'
    from='romeo@montague.net/orchard'
    to='juliet@capulet.com/balcony'
    id='disco1'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

Listing 10: Service Discovery information response

```
<iq type='result'
    from='juliet@capulet.com/balcony'
    to='romeo@montague.net/orchard'
    id='disco1'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    ...
    <feature var='jabber:x:data'/>
```

```
    ...
  </query>
</iq>
```

If an entity supports data forms indirectly through inclusion of data forms in a wrapper namespace (rather than directly within a <message/> stanza), it MUST NOT advertise support for the 'jabber:x:data' namespace, since support is implicit in support for the wrapper protocol.

7 Security Considerations

There are no security concerns related to this specification above and beyond those described in the relevant section of XMPP Core.

8 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA) [12].

9 XMPP Registrar Considerations

9.1 Protocol Namespaces

The XMPP Registrar [13] includes the 'jabber:x:data' namespace in its registry of protocol namespaces.

9.2 Parameter Values

The XMPP Registrar maintains a registry of parameter values related to the 'jabber:x:data' namespace, specifically as defined in Field Standardization for Data Forms (XEP-0068) [14]; the registry is located at <http://xmpp.org/registrar/formtypes.html>.

[12] The Internet Assigned Numbers Authority (IANA) is the central coordinator for the assignment of unique parameter values for Internet protocols, such as port numbers and URI schemes. For further information, see <http://www.iana.org/>.

[13] The XMPP Registrar maintains a list of reserved protocol namespaces as well as registries of parameters used in the context of XMPP extension protocols approved by the XMPP Standards Foundation. For further information, see <http://xmpp.org/registrar/>.

[14] XEP-0068: Field Data Standardization for Data Forms <http://xmpp.org/extensions/xep-0068.html>.

10 XML Schema

This schema is descriptive, not normative.

```
<?xml version='1.0' encoding='UTF-8'?>

<xs:schema
    xmlns:xs='http://www.w3.org/2001/XMLSchema'
    targetNamespace='jabber:x:data'
    xmlns='jabber:x:data'
    elementFormDefault='qualified'>

  <xs:annotation>
    <xs:documentation>
      The protocol documented by this schema is defined in
      XEP-0004: http://www.xmpp.org/extensions/xep-0004.html
    </xs:documentation>
  </xs:annotation>

  <xs:element name='x'>
    <xs:complexType>
      <xs:sequence>
        <xs:element name='instructions'
                    minOccurs='0'
                    maxOccurs='unbounded'
                    type='xs:string'/>
        <xs:element name='title' minOccurs='0' type='xs:string'/>
        <xs:element ref='field' minOccurs='0' maxOccurs='unbounded'/>
        <xs:element ref='reported' minOccurs='0' maxOccurs='1'/>
        <xs:element ref='item' minOccurs='0' maxOccurs='unbounded'/>
      </xs:sequence>
      <xs:attribute name='type' use='required'>
        <xs:simpleType>
          <xs:restriction base='xs:NCName'>
            <xs:enumeration value='cancel'/>
            <xs:enumeration value='form'/>
            <xs:enumeration value='result'/>
            <xs:enumeration value='submit'/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>

  <xs:element name='field'>
    <xs:complexType>
      <xs:sequence>
        <xs:element name='desc' minOccurs='0' type='xs:string'/>
        <xs:element name='required' minOccurs='0' type='empty'/>
```

```
      <xs:element ref='value' minOccurs='0' maxOccurs='unbounded'/>
      <xs:element ref='option' minOccurs='0' maxOccurs='unbounded'/>
    </xs:sequence>
    <xs:attribute name='label' type='xs:string' use='optional'/>
    <xs:attribute name='type' use='optional'>
      <xs:simpleType>
        <xs:restriction base='xs:NCName'>
          <xs:enumeration value='boolean'/>
          <xs:enumeration value='fixed'/>
          <xs:enumeration value='hidden'/>
          <xs:enumeration value='jid-multi'/>
          <xs:enumeration value='jid-single'/>
          <xs:enumeration value='list-multi'/>
          <xs:enumeration value='list-single'/>
          <xs:enumeration value='text-multi'/>
          <xs:enumeration value='text-private'/>
          <xs:enumeration value='text-single'/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name='var' type='xs:string' use='optional'/>
  </xs:complexType>
</xs:element>

<xs:element name='option'>
  <xs:complexType>
    <xs:sequence>
      <xs:element ref='value'/>
    </xs:sequence>
    <xs:attribute name='label' type='xs:string' use='optional'/>
  </xs:complexType>
</xs:element>

<xs:element name='value' type='xs:string'/>

<xs:element name='reported'>
  <xs:annotation>
    <xs:documentation>
      When contained in a "reported" element, the "field" element
      SHOULD NOT contain a "value" child.
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element ref='field' maxOccurs='unbounded'/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
  <xs:element name='item'>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref='field' maxOccurs='unbounded'/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:simpleType name='empty'>
    <xs:restriction base='xs:string'>
      <xs:enumeration value=''/>
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```

11 Changes in Final State

The following substantive protocol changes have been made while this specification has been in the Final state:

- Specified that the 'var' attribute is required for all field types except "fixed", for which the 'var' attribute is optional.
- Specified when to advertise support via service discovery.
- Removed references to <presence/> stanzas.

12 Changes in Draft State

The following substantive protocol changes were made while this specification was in the Draft state:

- The <x/> element MAY be included directly in <message/> and <presence/> stanzas.
- The <x/> element MAY contain a <title/> child for forms and results.
- The <x/> element MUST possess a 'type' attribute.
- A <field/> element MAY be of type='jid-single'.
- Results MAY be reported back in <item/> tags.
- Results MAY contain a <reported/> element with result set.
- The <reported/> fields MAY possess a 'type' attribute to provide hints about how to interact with the data (type='jid-single' being the most useful).