XEP-0050: Ad-Hoc Commands

Matthew Miller
mailto:linuxwolf@outer-planes.net
xmpp:linuxwolf@outer-planes.net

2020-06-09
Version 1.3.0

Status	Type	Short Name
Draft	Standards Track	commands

This document defines an XMPP protocol extension for advertising and executing application-specific commands, such as those related to a configuration workflow. Typically the commands contain data forms (XEP-0004) in order to structure the information exchange.

Legal

Copyright

This XMPP Extension Protocol is copyright © 1999 – 2024 by the XMPP Standards Foundation (XSF).

Permissions

Permission is hereby granted, free of charge, to any person obtaining a copy of this specification (the "Specification"), to make use of the Specification without restriction, including without limitation the rights to implement the Specification in a software program, deploy the Specification in a network service, and copy, modify, merge, publish, translate, distribute, sublicense, or sell copies of the Specification, and to permit persons to whom the Specification is furnished to do so, subject to the condition that the foregoing copyright notice and this permission notice shall be included in all copies or substantial portions of the Specification. Unless separate permission is granted, modified works that are redistributed shall not contain misleading information regarding the authors, title, number, or publisher of the Specification, and shall not claim endorsement of the modified works by the authors, any organization or project to which the authors belong, or the XMPP Standards Foundation.

Warranty

NOTE WELL: This Specification is provided on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE.

Liability

In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall the XMPP Standards Foundation or any author of this Specification be liable for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising from, out of, or in connection with the Specification or the implementation, deployment, or other use of the Specification (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if the XMPP Standards Foundation or such author has been advised of the possibility of such damages.

Conformance

This XMPP Extension Protocol has been contributed in full conformance with the XSF's Intellectual Property Rights Policy (a copy of which can be found at <https://xmpp.org/about/xsf/ipr-policy> or obtained by writing to XMPP Standards Foundation, P.O. Box 787, Parker, CO 80134 USA).

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Concepts . . . 1
- 1.3 Prerequisites . . . 1

2 Use Cases **2**
- 2.1 Discovering Support . . . 2
- 2.2 Retrieving the Command List . . . 2
- 2.3 Announcing the Command List . . . 4
- 2.4 Executing Commands . . . 5
  - 2.4.1 Simple Execution . . . 5
  - 2.4.2 Multiple Stages . . . 6
  - 2.4.3 Canceling . . . 10

3 Implementation Notes **11**
- 3.1 Defined/Required Command Nodes . . . 11
- 3.2 Command Nodes . . . 11
- 3.3 Session Lifetime . . . 11
- 3.4 Command Actions . . . 11
- 3.5 Command Payloads . . . 12
  - 3.5.1 Use of Data Forms . . . 13
- 3.6 Commands Successful/Failed . . . 13
- 3.7 Internationalization and Localization . . . 13

4 Formal Description **15**
- 4.1 <command/> Element . . . 15
- 4.2 <actions/> Element . . . 16
- 4.3 <note/> Element . . . 17
- 4.4 Possible Errors . . . 17

5 Security Considerations **18**

6 IANA Considerations **19**

7 XMPP Registrar Considerations **19**
- 7.1 Protocol Namespaces . . . 19
- 7.2 Service Discovery Identities . . . 19
- 7.3 Well-Known Service Discovery Nodes . . . 20
- 7.4 URI Query Types . . . 20

8 XML Schema **21**

9 Acknowledgements **24**

1 Introduction

This document specifies an XMPP protocol extension that enables an entity to initiate a command session where there is no preferred namespace. It also specifies a protocol extension for describing the types of ad hoc sessions, similar in concept to a menu.

1.1 Motivation

The motivation for such a protocol comes from the desire to expand Jabber technologies outside the domain of instant messaging. Similar to web applications, these "Jabber applications" are systems in which, via a compliant Jabber client, a user (or automated process) can interact with the application. The client need not be specially-written in order to take advantage of this Jabber application.
This mechanism allows for a larger base of Jabber entities to participate as part of larger application architectures. Although specialized clients would be preferred in many environments, this protocol allows for applications to have a wider audience (i.e., any compliant Jabber client).

1.2 Concepts

The namespace governing this protocol is "http://jabber.org/protocol/commands" (hereafter referred to as x-commands). This namespace relies on the <iq/> element for execution, and can use the <message/> element for announcing command lists. This protocol depends on Service Discovery (XEP-0030) [1] for reporting and announcing command lists. This namespace is intended to complement Data Forms (XEP-0004) [2] (jabber:x:data), but is not necessarily dependent upon it.

1.3 Prerequisites

Support of x-commands implies support for "jabber:x:data" (although this requirement may be replaced and/or amended with a requirement to support Feature Negotiation (XEP-0020) [3] by performing the appropriate negotations before executing commands). x-commands provides a bootstrap for performing ad-hoc "jabber:x:data" processes, while the data itself is conveyed using "jabber:x:data".
The x-commands namespace is not designed to replace machine-to-machine oriented RPC systems such as Jabber-RPC (XEP-0009) [4], where the two entities fully understand the command's purpose and behavior prior to execution. x-commands is oriented more for human interaction, where the user agent (such as a compliant Jabber client) most likely has no prior

[1] XEP-0030: Service Discovery <https://xmpp.org/extensions/xep-0030.html>.
[2] XEP-0004: Data Forms <https://xmpp.org/extensions/xep-0004.html>.
[3] XEP-0020: Feature Negotiation <https://xmpp.org/extensions/xep-0020.html>.
[4] XEP-0009: Jabber-RPC <https://xmpp.org/extensions/xep-0009.html>.

knowledge of the command's purpose and behavior.

2 Use Cases

2.1 Discovering Support

To determine if an entity supports x-commands, the requester uses Service Discovery. The requester makes an "#info" query to the responder. If supported, the responder includes a <feature/> with the "var" of "http://jabber.org/protocol/commands".

Listing 1: Disco request for information

```
<iq type='get'
    to='responder@domain'
    from='requester@domain'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

Listing 2: Disco result for information

```
<iq type='result'
    from='responder@domain'
    to='requester@domain'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    ...
    <feature var='http://jabber.org/protocol/commands'/>
    ...
  </query>
</iq>
```

2.2 Retrieving the Command List

To find what commands an entity provides, the requester uses Service Discovery. Each command is a node of the responder, under the fixed node "http://jabber.org/protocol/commands" (for which the service discovery identity category is "automation" and type is "command-list"). Use of a fixed node for all commands of an entity allows for immediate retrieval of commands.
Each command is a disco item. The node attribute of <item/> identifies the command, and the name attribute is the label for the command.
The requester retrieves the list of commands by querying for the responder's items for the node "http://jabber.org/protocol/commands":

Listing 3: Disco request for items

```
<iq type='get'
    from='requester@domain'
    to='responder@domain'>
  <query xmlns='http://jabber.org/protocol/disco#items'
         node='http://jabber.org/protocol/commands'/>
</iq>
```

Listing 4: Disco result for items

```
<iq type='result'
    to='requester@domain'
    from='responder@domain'>
  <query xmlns='http://jabber.org/protocol/disco#items'
         node='http://jabber.org/protocol/commands'>
    <item jid='responder@domain'
          node='list'
          name='List␣Service␣Configurations'/>
    <item jid='responder@domain'
          node='config'
          name='Configure␣Service'/>
    <item jid='responder@domain'
          node='reset'
          name='Reset␣Service␣Configuration'/>
    <item jid='responder@domain'
          node='start'
          name='Start␣Service'/>
    <item jid='responder@domain'
          node='stop'
          name='Stop␣Service'/>
    <item jid='responder@domain'
          node='restart'
          name='Restart␣Service'/>
  </query>
</iq>
```

The result can then be used by the client to populate a menu, a dialog of buttons, or whatever is appropriate to the current user interface. The responder is not required to send the same list of commands to all requesters.
If additional information about a command is desired, the requester queries for disco information on the command node:

Listing 5: Disco request for command information

```
<iq type='get'
    from='requester@domain'
    to='responder@domain'>
  <query xmlns='http://jabber.org/protocol/disco#info'
         node='config'/>
</iq>
```

Listing 6: Disco result for command information

```
<iq type='result'
    to='requester@domain'
    from='responder@domain'>
  <query xmlns='http://jabber.org/protocol/disco#info'
         node='config'>
    <identity name='Configure␣Service'
              category='automation'
              type='command-node'/>
    <feature var='http://jabber.org/protocol/commands'/>
    <feature var='jabber:x:data'/>
  </query>
</iq>
```

A responder MUST at least provide <identity category='automation' type='command-node'/> and <feature var='http://jabber.org/protocol/commands'/>, and SHOULD include <feature var='jabber:x:data'/>. It is not required to support additional information about a command. If the command is not available to the requester, the responder SHOULD respond with a 403 "Forbidden" error.

2.3 Announcing the Command List

In some cases, a responder entity may find it appropriate to automatically push this information (e.g. a subscribed entity becomes available). In this case, the entity sends a <message/> containing the proper disco#items <query/>:

Listing 7: Announcing commands (via <message/>)

```
<message from='responder@domain' to='requester@domain'>
  <subject>Service Controls</subject>
  <query xmlns='http://jabber.org/protocol/disco#items'
         node='http://jabber.org/protocol/commands'>
    <item jid='responder@domain'
          node='list'
          name='List␣Service␣Configurations'/>
    <item jid='responder@domain'
          node='config'
          name='Configure␣Service'/>
    <item jid='responder@domain'
          node='reset'
          name='Reset␣Service␣Configuration'/>
    <item jid='responder@domain'
          node='start'
          name='Start␣Service'/>
    <item jid='responder@domain'
          node='stop'
          name='Stop␣Service'/>
```

```
    <item jid='responder@domain'
          node='restart'
          name='Restart␣Service'/>
  </query>
</message>
```

The only portion required is <query xmlns='http://jabber.org/protocol/disco#items'/>. Any other information (such as the <subject/> in the foregoing example) is OPTIONAL.

2.4 Executing Commands

2.4.1 Simple Execution

To execute a command, the requester sends an <iq/> containing the command to execute:

Listing 8: Execute command request

```
<iq type='set' to='responder@domain' id='exec1'>
  <command xmlns='http://jabber.org/protocol/commands'
           node='list'
           action='execute'/>
</iq>
```

The requester MAY include the "action='execute'", although this is implied.
If the command does not require any user interaction (returns results only), the responder sends a packet similar to the following:

Listing 9: Execute command result

```
<iq type='result' from='responder@domain' to='requester@domain' id='
    exec1'>
  <command xmlns='http://jabber.org/protocol/commands'
           sessionid='list:20020923T213616Z-700'
           node='list'
           status='completed'>
    <x xmlns='jabber:x:data' type='result'>
      <title>Available Services</title>
      <reported>
        <field var='service' label='Service'/>
        <field var='runlevel-1' label='Single-User␣mode'/>
        <field var='runlevel-2' label='Non-Networked␣Multi-User␣mode'/
            >
        <field var='runlevel-3' label='Full␣Multi-User␣mode'/>
        <field var='runlevel-5' label='X-Window␣mode'/>
      </reported>
      <item>
        <field var='service'><value>httpd</value></field>
        <field var='runlevel-1'><value>off</value></field>
```

```
        <field var='runlevel-2'><value>off</value></field>
        <field var='runlevel-3'><value>on</value></field>
        <field var='runlevel-5'><value>on</value></field>
      </item>
      <item>
        <field var='service'><value>postgresql</value></field>
        <field var='runlevel-1'><value>off</value></field>
        <field var='runlevel-2'><value>off</value></field>
        <field var='runlevel-3'><value>on</value></field>
        <field var='runlevel-5'><value>on</value></field>
      </item>
      <item>
        <field var='service'><value>jabberd</value></field>
        <field var='runlevel-1'><value>off</value></field>
        <field var='runlevel-2'><value>off</value></field>
        <field var='runlevel-3'><value>on</value></field>
        <field var='runlevel-5'><value>on</value></field>
      </item>
    </x>
  </command>
</iq>
```

The above example shows the command execution resulting in a "jabber:x:data" form. It is also possible that one or more URLs (specified via Out-of-Band Data (XEP-0066) [5]) could be returned.

2.4.2 Multiple Stages

If the command requires more interaction, the responder sends a result <iq/> that contains the command information and the form to be filled out:

Listing 10: Execute command request (stage 1)

```
<iq type='set' to='responder@domain' id='exec1'>
  <command xmlns='http://jabber.org/protocol/commands'
           node='config'
           action='execute'/>
</iq>
```

Listing 11: Execute command result (stage 1)

```
<iq type='result' from='responder@domain' to='requester@domain' id='
    exec1'>
  <command xmlns='http://jabber.org/protocol/commands'
           sessionid='config:20020923T213616Z-700'
           node='config'
```

[5] XEP-0066: Out of Band Data <https://xmpp.org/extensions/xep-0066.html>.

```
          status='executing'>
    <actions execute='next'>
      <next/>
    </actions>
    <x xmlns='jabber:x:data' type='form'>
      <title>Configure Service</title>
      <instructions>
        Please select the service to configure.
      </instructions>
      <field var='service' label='Service' type='list-single'>
        <option><value>httpd</value></option>
        <option><value>jabberd</value></option>
        <option><value>postgresql</value></option>
      </field>
    </x>
  </command>
</iq>
```

The <command/> SHOULD include an <actions/> element, which specifies the details of what the allowed actions are for this stage of execution. Each element within <action/> matches a possible value for the <command/> element's "action" attribute. The "execute" attribute defines which of the included actions is considered the equivalent to "execute" for this stage. In the above example, the only allowed action is to progress to the next stage, which is also the default.

Note: The "execute" action (not the attribute) is ambiguous and may have undefined behaviour in some implementations. Relying on the specific behavior of the execute action is discouraged, especially in cases where the "complete" action is not allowed and the "execute" attribute is not specified (see also Command Actions.

The requester then submits the form, maintaining the command node and sessionid:

Listing 12: Execute command request (stage 2)

```
<iq type='set' to='responder@domain' id='exec2'>
  <command xmlns='http://jabber.org/protocol/commands'
           sessionid='config:20020923T213616Z-700'
           node='config'>
    <x xmlns='jabber:x:data' type='submit'>
      <field var='service'>
        <value>httpd</value>
      </field>
    </x>
  </command>
</iq>
```

The responder then provides the next stage's form in the result [6]:

Listing 13: Execute command result (stage 2)

```
<iq type='result' from='responder@domain' to='requester@domain' id='
    exec2'>
  <command xmlns='http://jabber.org/protocol/commands'
           sessionid='config:20020923T213616Z-700'
           node='config'
           status='executing'>
    <actions execute='complete'>
      <prev/>
      <complete/>
    </actions>
    <x xmlns='jabber:x:data' type='form'>
      <title>Configure Service</title>
      <instructions>
        Please select the run modes and state for 'httpd'.
      </instructions>
      <field var='runlevel' label='Run␣Modes' type='list-multi'>
        <value>3</value>
        <value>5</value>
        <option label='Single-User'><value>1</value></option>
        <option label='Non-Networked␣Multi-User'><value>2</value></
            option>
        <option label='Full␣Multi-User'><value>3</value></option>
        <option label='X-Window'><value>5</value></option>
      </field>
      <field var='state' label='Run␣State' type='list-single'>
        <value>off</value>
        <option label='Active'><value>off</value></option>
        <option label='Inactive'><value>on</value></option>
      </field>
    </x>
  </command>
</iq>
```

The requester then submits the second stage's form, again maintaining the node and sessionid:

Listing 14: Execute command request (stage 3)

```
<iq type='set' to='responder@domain' id='exec3'>
  <command xmlns='http://jabber.org/protocol/commands'
           sessionid='config:20020923T213616Z-700'
           node='config'>
```

[6] Note that the second stage can be reverted to the first stage or completed (signaled by the inclusion of the <prev/> and <complete/> elements), and that the default action is to complete execution (signaled by the "execute" attribute's value of "complete").

```
    <x xmlns='jabber:x:data' type='submit'>
      <field var='runlevel'>
        <value>3</value>
      </field>
      <field var='state'>
        <value>on</value>
      </field>
    </x>
  </command>
</iq>
```

Listing 15: Execute command result (stage 3)

```
<iq type='result' from='responder@domain' to='requester@domain' id='
    exec3'>
  <command xmlns='http://jabber.org/protocol/commands'
           sessionid='config:20020923T213616Z-700'
           node='config'
           status='completed'>
    <note type='info'>Service 'httpd' has been configured.</note>
  </command>
</iq>
```

If the requester wishes to revert to the previous stage, it sends an <iq/> with the command's node and sessionid, and "action='prev'":

Listing 16: Execute command request (revert from stage 2 to stage 1)

```
<iq type='set' to='responder@domain' id='exec2'>
  <command xmlns='http://jabber.org/protocol/commands'
           sessionid='config:20020923T213616Z-700'
           node='config'
           action='prev'/>
</iq>
```

If the responder accepts this, it responds with the previous stage's command [7]:

Listing 17: Execute command result (revert from stage 2 to stage 1)

```
<iq type='result' from='responder@domain' to='requester@domain' id='
    exec2'>
  <command xmlns='http://jabber.org/protocol/commands'
           sessionid='config:20020923T213616Z-700'
           node='config'
           status='executing'>
    <actions execute='next'>
      <next/>
```

[7] The responder MAY present "remembered" field values, but doing so is OPTIONAL.

```
    </actions>
    <x xmlns='jabber:x:data' type='form'>
      <title>Configure Service</title>
      <instructions>
        Please select the service to configure.
      </instructions>
      <field var='service' label='Service' type='list-single'>
        <value>httpd</value>
        <option><value>httpd</value></option>
        <option><value>jabberd</value></option>
        <option><value>postgresql</value></option>
      </field>
    </x>
  </command>
</iq>
```

2.4.3 Canceling

In the case where a command has multiple stages, the requester may wish to cancel at some point. To cancel, the requester sends the continuing command request with an "action='cancel'":

Listing 18: Execute command request (stage 2; canceling)

```
<iq type='set' to='responder@domain' id='exec3'>
  <command xmlns='http://jabber.org/protocol/commands'
           sessionid='config:20020923T213616Z-700'
           node='config'
           action='cancel'/>
</iq>
```

This enables the responder to free any resources allocated during the process. The responder MUST reply with the success of the command:

Listing 19: Execute command result (stage 2; canceled)

```
<iq type='result' from='responder@domain' to='requester@domain' id='
    exec3'>
  <command xmlns='http://jabber.org/protocol/commands'
           sessionid='config:20020923T213616Z-700'
           node='config'
           status='canceled'/>
</iq>
```

3 Implementation Notes

3.1 Defined/Required Command Nodes

All commands used in the above examples are for illustrative purposes only. There are no predefined or required commands.

3.2 Command Nodes

Each command is identified by its 'node' attribute. This matches the 'node' attribute from the service discovery <item/> element. Service Discovery requires that all 'node' values be unique within a given JID. This document requires that the 'node' value used in <command/> exactly match the value used in the <item/> element. It is the responsibility of the responder implementation to ensure each command's node is unique for their JID.

3.3 Session Lifetime

The execution of a command exists within the concept of a session. Each session is identified by the 'sessionid' attribute, and SHOULD be valid only between one requester/responder pair. The responder is responsible for determining the session lifetime, with some help from the requester.
The requester starts a new session for a command by simply sending a <command/> with the 'node' attribute (and optionally the 'status' attribute with a value of "execute"). Once the 'sessionid' attribute is given to the requester, it is the requester's responsibility to maintain it for the session's lifetime. A session ends when the responder sends a <command status='completed'/> or the requester sends a <command action='cancel'/> with the provided 'sessionid' value.
Once a session has ended, its 'sessionid' value SHOULD NOT be used again. It is the responder's responsibility to ensure that each 'sessionid' value is unique.
It may be possible for a requester to be executing more than one session of the same command with a given responder. If the responder does not allow more than one session of the same command with the same requester, the responder MUST return a <not-allowed/> error (see Error Condition Mappings (XEP-0086) [8]).

3.4 Command Actions

The result for each stage (other than the last) of a command's execution SHOULD include an <actions/> element. The user-agent can use this information to present a more-intelligent user interface, such as a "druid" or "wizard".
For a user-agent, a typical interpretation of the <actions/> information (or lack thereof) would

[8] XEP-0086: Error Condition Mappings <https://xmpp.org/extensions/xep-0086.html>.

be the following:

1. The action ”cancel” is always allowed.
2. If there is no <actions/> element, the user-agent can use a single-stage dialog or view.
   - The action ”execute” is equivalent to the action ”complete”.
3. If there is an <actions/> element, the user-agent usually uses a multi-stage dialog or view, such as a wizard.
   - The action ”execute” is equivalent to the action specified in the ”execute” attribute. If the ”execute” attribute is absent, it defaults to ”next”. A form which has an <actions/> element and an ”execute” attribute which evaluates (taking the default into account if absent) to an action which is not allowed is therefore invalid.
   - The ”prev” action is typically the ”back” or ”previous” button or option in a wizard. If <prev/> is not contained by the <actions/>, it is disabled.
   - The ”next” action is typically the ”next” button or option in a wizard. If <next/> is not contained by the <actions/>, it is disabled.
   - The ”complete” action is typically the ”finish” or ”done” button or option in a wizard. If <complete/> is not contained by the <actions/>, it is disabled.
   - If the <actions/> possesses the ”execute” attribute, that value is the default button or option. If the <actions/> does not possess the ”execute” attribute, there is no default button or option.
   - As the ”execute” command is equivalent to another command (”next” in the absence of an ”execute” attribute), it is not suggested that a user-agent shows both the ”execute” and the command that ”execute” is an alias of in their interface.

Responders SHOULD use the following guidelines when providing <actions/>:

- The ”execute” attribute SHOULD NOT specify a value that does not match one of the allowed actions.
- As user-agents treat the absence of an ”execute” attribute as equivalent to ”next”, when the <actions/> element is provided, it is invalid to return a command that has no ”execute” attribute, an actions element, and no <next/> action

3.5 Command Payloads

On its own, the <command/> has very little usefulness. It relies on its payload to give full meaning to its use. The payload can be elements in any namespace that makes sense and is understood (such as ”jabber:x:data”), and/or one or more <note/> elements. Any namespaced elements can be used within a <command/>. The only limitations are that the elements not require certain parent elements (such as <iq/>), or specifically allow for <command/>

qualified by the "http://jabber.org/protocol/commands" namespace as a possible parent element.
As a general rule, the payload is provided only by the responder. The primary exception to this rule is with the "jabber:x:data" extension (and other namespaces with similar semantics). In this case, if the responder provides a form to submit, the requester SHOULD respond with the submitted data (using the semantics from XEP-0004).
When the precedence of these payload elements becomes important (such as when both "jabber:x:data" and "jabber:x:oob" elements are present), the order of the elements SHOULD be used. Those elements that come earlier in the child list take precedence over those later in the child list. The requester SHOULD consider those elements qualified by the same namespace as having an equivalent precedence (such as if multiple "jabber:x:oob" elements are included).

3.5.1 Use of Data Forms

When the payload is "jabber:x:data", there are certain conditions applied. The requester SHOULD NOT use a "jabber:x:data" type other than "submit". Responders SHOULD consider any <x type='cancel'/> to be <x type='submit'/>.

3.6 Commands Successful/Failed

The status of command execution signals only if the command is executing, has been completed, or been canceled. If completed, the "status" attribute does not specify if it completed successfully or not. If a command completes but fails, the responder MUST include at least one <note type='error'/> with the <command status='completed'/> it returns.

3.7 Internationalization and Localization

The requester SHOULD provide its locale information using the "xml:lang" attribute on either the <iq/> (RECOMMENDED) or <command/> element. Each execution session (identified by the "sessionid" attribute) SHOULD use only one language/locale, and requesters and responders SHOULD assume the first language/locale specified applies. The responder SHOULD specify the language/locale with the every command session's response.

Listing 20: Execute command request (with language/locale)

```
<iq type='set' to='responder@domain' id='exec1' xml:lang='en-us'>
  <command xmlns='http://jabber.org/protocol/commands'
           node='list'
           action='execute'/>
</iq>
```

Listing 21: Execute command result (with language/locale)

```
<iq type='result'
    from='responder@domain'
    to='requester@domain'
    id='exec1'
    xml:lang='en-us'>
  <command xmlns='http://jabber.org/protocol/commands'
           sessionid='list:20020923T213616Z-700'
           node='list'
           status='completed'>
    <x xmlns='jabber:x:data' type='result'>
      <title>Available Services</title>
      <reported>
        <field var='service' label='Service'/>
        <field var='runlevel-1' label='Single-User␣mode'/>
        <field var='runlevel-2' label='Non-Networked␣Multi-User␣mode'/
            >
        <field var='runlevel-3' label='Full␣Mult-User␣mode'/>
        <field var='runlevel-5' label='X-Window␣mode'/>
      </reported>
      <item>
        <field var='service'><value>httpd</value></field>
        <field var='runlevel-1'><value>off</value></field>
        <field var='runlevel-2'><value>off</value></field>
        <field var='runlevel-3'><value>on</value></field>
        <field var='runlevel-5'><value>on</value></field>
      </item>
      <item>
        <field var='service'><value>postgresql</value></field>
        <field var='runlevel-1'><value>off</value></field>
        <field var='runlevel-2'><value>off</value></field>
        <field var='runlevel-3'><value>on</value></field>
        <field var='runlevel-5'><value>on</value></field>
      </item>
      <item>
        <field var='service'><value>jabberd</value></field>
        <field var='runlevel-1'><value>off</value></field>
        <field var='runlevel-2'><value>off</value></field>
        <field var='runlevel-3'><value>on</value></field>
        <field var='runlevel-5'><value>on</value></field>
      </item>
    </x>
  </command>
</iq>
```

Within the "http://jabber.org/protocol/commands" schema, the language/locale applies only to the human-readable character data for <info/> elements. It SHOULD also apply to all payload elements, appropriate to their respective specifications.
Responders MUST take this into consideration, and properly account for the language/locale

settings within payloads. If the responder cannot accomodate the requested language/locale, it SHOULD respond with a <bad-request/> (<bad-locale/>) error condition.

Listing 22: Execute command request (with language/locale)

```
<iq type='set' to='responder@domain' id='exec1' xml:lang='fr-ca'>
  <command xmlns='http://jabber.org/protocol/commands'
           node='list'
           action='execute'/>
</iq>
```

Listing 23: Execute command failed result (with language/locale)

```
<iq type='error' from='responder@domain' to='requester@domain/resource
    ' id='exec1'>
  <command xmlns='http://jabber.org/protocol/commands'
           node='list'
           action='execute'
           xml:lang='fr-ca'/>
  <error type='modify' code='400'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <bad-locale xmlns='http://jabber.org/protocol/commands'/>
  </error>
</iq>
```

4 Formal Description

The focal element in x-commands is <command/>. It is the element used to guide the process, and the element used to report command options.

4.1 <command/> Element

Each <command/> contains attributes for a node, a "session id", an action type, a status type, and a language/locale specifier. A command MAY contain zero or more <note/> elements and MAY contain other namespaced elements as payload. Elements qualified by the "jabber:x:data" and "jabber:x:oob" namespaces are the typical payload.
The "node" attribute uniquely identifies the command. This attribute MUST be present.
The "sessionid" attribute helps to track a command execution across multiple stages. This attribute MUST be present for subsequent stages, and the responder SHOULD initialize (if not provided) or maintain this attribute. The value of this attribute MUST NOT be empty or null, but otherwise can be any string value. This value MUST be maintained by a requester while executing a command.
The "status" attribute describes the current status of this command. This value SHOULD be set only by the responder. If specified by the requester, the responder MUST ignore it. The

value of ”status” MUST be one of the following:

Status	Description
executing	The command is being executed.
completed	The command has completed. The command session has ended.
canceled	The command has been canceled. The command session has ended.

The ”action” attribute specifies the action to undertake with the given command. This value SHOULD be set only by the requester. If specified by the responder, the requester MUST ignore it. The value of ”action” MUST be one of the following:

Action	Description
execute	The command should be executed or continue to be executed. This is the default value.
cancel	The command should be canceled.
prev	The command should be digress to the previous stage of execution.
next	The command should progress to the next stage of execution.
complete	The command should be completed (if possible).

The ”xml:lang” attribute specifies the language/locale this <command/> is intended for. This attribute MAY be specified by the requester to request a specific language/locale, and SHOULD be included by the responder to indicate the language/locale in use.
The children of a <command/> element (other than <actions/> and <note/>) pertain to the command’s execution. The order of these elements denote their precedence, so that those elements earlier in the list have higher precedence.

4.2 <actions/> Element

The allowed actions for a particular stage of execution are provided by the <actions/> element. This element SHOULD be provided by the responder if the command’s execution is not complete, and SHOULD NOT ever be provided by the requester. It contains a single attribute to specify what the ”execute” action equals. It contains child elements to specify what the allowed actions are.
The ”execute” attribute specifies what the action ”execute” is equivalent to. In user-agent interfaces, this represents the default behavior. This attribute MAY be specified by the responder, and MUST equal one of the ”action” attribute values for <command/>. The value of this attribute SHOULD match the local name of one of the contained child elements.

The child elements contained by <action/> specify the allowed actions. The name of each child element MUST be one of the following:

- prev
- next
- complete

4.3 <note/> Element

Notes about the current status of commands are provided by <note/> elements. This element contains information about current conditions in a command sequence. This element has an attribute that defines the type of note. The body of a <note/> should contain a user-readable text message.
The "type" attribute specifies the severity of the note. This attribute is OPTIONAL, and implies "info" if not present. The value of this attribute MUST be one of the following:

Type	Description
info	The note is informational only. This is not really an exceptional condition.
warn	The note indicates a warning. Possibly due to illogical (yet valid) data.
error	The note indicates an error. The text should indicate the reason for the error.

4.4 Possible Errors

To simplify the discussion on error conditions, this document uses the following mapping between namespace URIs and namespace prefixes [9]:

Prefix	URI
xmpp	urn:ietf:params:xml:ns:xmpp-stanzas
cmd	http://jabber.org/protocol/commands

Below are the possible errors that can occur during execution.

[9] This mapping is provided solely for the purpose of simplifying this discussion.

Error Type	General Condition	Specific Condition	Description
modify	<xmpp:bad-request/>	<cmd:malformed-action/>	The responding JID does not understand the specified action.
modify	<xmpp:bad-request/>	<cmd:bad-action/>	The responding JID cannot accept the specified action.
modify	<xmpp:bad-request/>	<cmd:bad-locale/>	The responding JID cannot accept the specified language/locale.
modify	<xmpp:bad-request/>	<cmd:bad-payload/>	The responding JID cannot accept the specified payload (e.g. the data form did not provide one or more required fields).
modify	<xmpp:bad-request/>	<cmd:bad-sessionid/>	The responding JID cannot accept the specified sessionid.
cancel	<xmpp:not-allowed/>	<cmd:session-expired/>	The requesting JID specified a sessionid that is no longer active (either because it was completed, canceled, or timed out).
cancel	<xmpp:forbidden/>	NONE	The requesting JID is not allowed to execute the command.
cancel	<xmpp:item-not-found/>	NONE	The responding JID cannot find the requested command node.
cancel	<xmpp:feature-not-implemented/>	NONE	The responding JID does not support ”http://jabber.org/protocol/commands”.

5 Security Considerations

Determining when a command can be executed based on permissions or rights is considered outside the scope of this document. Although such mechanisms are considered specific to the application and/or implementation of this document, future specifications may address these concerns.

When processing reported commands, the requester SHOULD consider any command node that does not match the JID of the responder to be suspicious, and ignore those command

nodes. Responders MUST report their own command nodes only, and not the command nodes of other entities. This can help prevent limited cases of spoofing and "social engineering".

6 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA) [10].

7 XMPP Registrar Considerations

7.1 Protocol Namespaces

The XMPP Registrar [11] includes 'http://jabber.org/protocol/commands' in its registry of protocol namespaces.

7.2 Service Discovery Identities

The XMPP Registrar includes "automation" in its registry of Service Discovery categories for use for any entities and nodes that provide automated or programmed interaction. This category has the following types:

Type	Description
command-list	The node for a list of commands; valid only for the node "http://jabber.org/protocol/commands".
command-node	A node for a specific command; the 'node' attribute uniquely identifies the command.

The registry submission is as follows:

```
<category>
    <name>automation</name>
    <desc>The "automation" category consists of entities and nodes
        that provide automated or programmed interaction.</desc>
```

[10] 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/>.

[11] 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 <https://xmpp.org/registrar/>.

```
    <type>
      <name>command-list</name>
      <desc>The node for a list of commands; valid only for the node "
          http://jabber.org/protocol/commands"</desc>
      <doc>XEP-0050</doc>
    </type>
    <type>
      <name>command-node</name>
      <desc>A node for a specific command; the 'node' attribute
          uniquely identifies the command</desc>
      <doc>XEP-0050</doc>
    </type>
  </category>
```

7.3 Well-Known Service Discovery Nodes

The XMPP Registrar includes "http://jabber.org/protocol/commands" in its registry of well-known Service Discovery nodes.

7.4 URI Query Types

As authorized by XMPP URI Query Components (XEP-0147) [12], the XMPP Registrar maintains a registry of queries and key-value pairs for use in XMPP URIs (see <https://xmpp.org/registrar/querytypes.html>).
The "command" querytype is defined herein for interaction with entities that support the ad-hoc command protocol, with keys of "action" and "node".

Listing 24: Command Action: IRI/URI

```
xmpp:montague.net?command;node=stats
```

Listing 25: Command Action: Resulting Stanza

```
<iq to='montague.net' type='set'>
  <command xmlns='http://jabber.org/protocol/commands' node='stats'/>
</iq>
```

The following submission registers the "command" querytype.

```
<querytype>
  <name>command</name>
  <proto>http://jabber.org/protocol/commands</proto>
  <desc>enables completion of ad-hoc commands</desc>
  <doc>XEP-0050</doc>
```

[12] XEP-0147: XMPP URI Query Components <https://xmpp.org/extensions/xep-0147.html>.

```
  <keys>
    <key>
      <name>action</name>
      <desc>the ad-hoc commands action type</desc>
      <values>
        <value>
          <name>cancel</name>
          <desc>a request to cancel processing of the command</desc>
        </value>
        <value>
          <name>complete</name>
          <desc>a request to complete processing of the command</desc>
        </value>
        <value>
          <name>execute</name>
          <desc>a request to execute the command (the default implied
              action)</desc>
        </value>
        <value>
          <name>next</name>
          <desc>a request to move to the next command in a series</
              desc>
        </value>
        <value>
          <name>prev</name>
          <desc>a request to move to the previous command in a series<
              /desc>
        </value>
      </values>
    </key>
    <key>
      <name>node</name>
      <desc>the command node</desc>
    </key>
  </keys>
</querytype>
```

8 XML Schema

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

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

```
<xs:annotation>
  <xs:documentation>
    The protocol documented by this schema is defined in
    XEP-0050: http://www.xmpp.org/extensions/xep-0050.html
  </xs:documentation>
</xs:annotation>

<xs:import namespace='http://www.w3.org/XML/1998/namespace'
           schemaLocation='http://www.w3.org/2001/03/xml.xsd'/>

<xs:element name='command'>
  <xs:complexType>
    <xs:choice>
      <xs:element ref='actions' minOccurs='0'/>
      <xs:element ref='note' minOccurs='0' maxOccurs='unbounded'/>
      <xs:any namespace='##other' minOccurs='0' maxOccurs='unbounded
         '/>
    </xs:choice>
    <xs:attribute name='action' use='optional'>
      <xs:simpleType>
        <xs:restriction base='xs:NCName'>
          <xs:enumeration value='cancel'/>
          <xs:enumeration value='complete'/>
          <xs:enumeration value='execute'/>
          <xs:enumeration value='next'/>
          <xs:enumeration value='prev'/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name='node' type='xs:string' use='required'/>
    <xs:attribute name='sessionid' type='xs:string' use='optional'/>
    <xs:attribute name='status' use='optional'>
      <xs:simpleType>
        <xs:restriction base='xs:NCName'>
          <xs:enumeration value='canceled'/>
          <xs:enumeration value='completed'/>
          <xs:enumeration value='executing'/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute ref='xml:lang' use='optional'/>
  </xs:complexType>
</xs:element>

<xs:element name='actions'>
  <xs:complexType>
    <xs:sequence>
      <xs:element name='prev' type='empty' minOccurs='0'/>
      <xs:element name='next' type='empty' minOccurs='0'/>
```

```
        <xs:element name='complete' type='empty' minOccurs='0'/>
      </xs:sequence>
      <xs:attribute name='execute' use='optional'>
        <xs:simpleType>
          <xs:restriction base='xs:NCName'>
            <xs:enumeration value='complete'/>
            <xs:enumeration value='next'/>
            <xs:enumeration value='prev'/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>

  <xs:element name='note'>
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base='xs:string'>
          <xs:attribute name='type' use='optional'>
            <xs:simpleType>
              <xs:restriction base='xs:NCName'>
                <xs:enumeration value='error'/>
                <xs:enumeration value='info'/>
                <xs:enumeration value='warn'/>
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>

  <xs:element name='bad-action' type='empty'/>
  <xs:element name='bad-locale' type='empty'/>
  <xs:element name='bad-payload' type='empty'/>
  <xs:element name='bad-sessionid' type='empty'/>
  <xs:element name='malformed-action' type='empty'/>
  <xs:element name='session-expired' type='empty'/>

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

</xs:schema>
```

9 Acknowledgements

Many thanks to Florian Schmaus, Christian Schudt and Anno van Vliet for their input and feedback on this specification.