TS-002
Tool Automation Harness

Abstract:	This XMPP extension describes a way in which any domain-specific tool can expose an automation harness via XMPP that allows other tools to control and/or monitor operations using that tool.

Authors:	Brian Bonnett, Les D'Souza, Kingston Duffie, Kenneth Green, Keith Kidd, Todd Law, Tom McBeath, Eric Miller, Madhusudan Nanjanagud, Kris Raney, Kate Xu

Status:	Final
Revision:	01
Revision date	June 2011
Submission:	ntaf ts-002

Legal Notices

This Specification has been created by the Network Test Automation Forum (NTAF). NTAF reserves the rights to at any time add to, amend, modify or withdraw all or any portion of this Specification.

Note: All parties who in any way intend to use this Specification for any purpose, are hereby put on notice that the possibility exists that practicing under this Specification may require the use of inventions covered by the patent rights held by third parties. By publication of this Specification NTAF makes no representation or warranty whatsoever, whether expressed or implied, that practicing under this will not infringe any third party rights, nor does NTAF make any representation or warranty whatsoever, whether expressed or implied, with respect to (1) any claim that has been or may be asserted by any third party, (2) the validity of any patent rights related to any such claim, (3) or the extent to which a license to use any such rights may or may not be available on reasonable and nondiscriminatory terms, or on any terms at all.

Revision History

Version	Date	By	Changes
01	2011-06-01		First release

Table of Contents

1. Introduction

This document describes an XMPP extension that allows an application or tool, with or without its own specialized man-machine user interface to expose an "automation harness" that allows that tool to be controlled and/or monitored by another tool via XMPP packet exchanges described here.

Motivation

Traditionally, XMPP has been used to facilitate immediate communication between humans. There are many XMPP client applications but most follow a familiar pattern allowing a human user to communicate with (and view presence of) other users.

However XMPP, because of its extensibility and XML orientation, is also suitable for human-to-machine, machine-to-human, and machine-to-machine communication as well. The widespread use of so-called "bots" is one example where a human-to-machine communication enables a human user to exchange information with a machine by sending it textual requests and, typically, receiving textual responses.

Consider a specialized computer program with a user interface designed to allow a human to perform certain specific tasks interactively. Now consider how one would incorporate those tasks into an automated process. Typically one would look for an API that allows equivalent functions to be performed, and would write programs or scripts to drive that API. The idea behind this extension is to enable tool developers to expose an "automation harness" on their tools that mirrors the high-level domain-specific actions taken at a user interface. In one mode, these tools can report user activity as a set of actions that can, in turn, be performed again later remotely via the same automation harness operating in an "automation" mode. Effectively, we are enabling domain-specific tools to interact with each other via XMPP recognizing that there may be human users behind either, both, or neither of the tools involved, depending on the situation.

Concept

At the heart of this extension is the notion of a "harness". This is a generic mechanism over XMPP whereby tools can advertise certain domain-specific command sets, providing all of the details about how these commands can be issued within the context of a session. The harness is fully self-describing, with an understanding that this is not just advertising an API that could be used by programmers, but may also be used to provide a user interface and therefore needs information in the description suitable for humans to use in participating in decisions about how commands will be invoked.

One will immediately note the parallels here to Ad Hoc Commands (XEP-0050) and to IO Data (XEP-0244). Ad Hoc Commands addresses a similar notion – where a set of commands are advertised and include information (in the form of Data Forms) suitable for a human user interface to be supported. IO Data takes this further to allow a much more extensible notion of input and output data – allowing any possible XML data structures in each case. While both of these extensions are relevant, they are deemed insufficient to the full need for sophisticated tools if they are to be fully automated, especially when human user interfaces are involved, and when a capture-replay model is sought. Many of the concepts from both of these extensions have been used to inform this new extension, while this extension does not specifically extend either of these.

Network Test Automation

The specific domain of interest to the authors is network test automation. In this world, it is common for a large set of both specialized and general-purpose tools to be assembled to create a complete solution. Traditionally, the implementation of such a solution usually involves a large amount of custom programming and is challenging as the tools involved are constantly evolving. Bilateral integrations between these various tools are prohibitively costly in terms of both time and labor. So the leaders in this market are seeking a better approach.

A new approach that is being used with some success is called "tool orchestration". This is the idea that rather than driving tools through APIs using scripts, one can define a high-level sequence of operations that may span a number of other tools through programmatic actions that correspond to human actions that would be performed

through the appropriate user interface for that tool. By combining this with a mechanism to capture human actions, one creates a rapid and robust solution.

Unfortunately, today, most tools have not anticipated this approach and therefore do not have "automation harnesses" built into them that allow user activity to be monitored and for similar action sequences to be requested remotely. But members of the Network Test Automation Forum have expressed interest in potentially adding such capabilities into the tools that they create. If all of these tools share a common mechanism for exposing this functionality, then we anticipate acceleration in innovation in this market that is desperately needed.

Consider the following typical network test environment. It consists of a defect tracking system, a test management system, a lab automation system, test automation scripting/authoring tools, test execution tools, a test scheduling system, a test reporting system, a variety of general purpose tools (like ANSI terminal emulators, web browsers, etc.), and various specialized network test tools such as traffic generators, network emulators, etc. A test authoring tool will create a script that will drive these various general-purpose and domain-specific test tools. The authoring tool will publish these tests/scripts into a test management system. A user may request that a certain set of tests be executed on a scheduled basis – causing the test scheduling system to request a reservation for the lab resources required by those tests. At the appropriate time, it will make a request of the test execution system to perform the actual testing which will, in turn, publish its results into the test reporting system. These results may be analyzed by a person or automatically, generating defect reports and/or updating the test management system accordingly. An automated tool might even decide to run additional tests based on the results of earlier tests. In this type of environment, we would like to have a consistent way for this wide variety of tools to integrate with one another without having to write a lot of specialized "glue" code. Instead, the capabilities of one tool could be consumed by other tools as needed. For example, a test scheduling system could use XMPP presence and discovery to find another tool that supports a harness that includes a set of reservation actions that it knows how to use. Several vendors may provide applicable solutions. And a test authoring tool can use a capture-replay mechanism spanning several other tools to document and then automate a testing process that requires the use of several other tools, and that can provide analysis functionality based on the self-describing nature of the information returned by those tools over these automation harnesses.

2. Use Cases

Discovering Harness Support

To determine if a given XMPP endpoint supports harnesses, one uses Service Discovery. If the entity supports harnesses, it will include this among its supported features.

Example 1. Disco request for features and corresponding response

```
<iq type='get' to='provider@domain/1' from='requester@domain/1' id='d1'>
   <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>

<iq type='result' from='provider@domain/1' to='requester@domain/1' id='d1'>
   <query xmlns='http://jabber.org/protocol/disco#info'>
    ...
      <feature var='http://ntaforum.org/2011/harness'/>
      <feature var='http://example.org/scp'/>
    ...
   </query>
</iq>
```

Discovering Harness Details

Through separate mechanisms not described here, a requester can find out what harnesses a given tool supports. For more information, see Tool Registration and Discovery specification. That requester may wish to learn more about a harness. To accomplish this, a requester issues a query-harness to the provider. The provider responds

with information describing the harness itself, the actions that can be performed on that harness (in the context of a session), and the events that the provider may issue during the lifetime of that session.

Example 3. Requesting details about a certain harness and the corresponding response

```
<iq type='get' to='provider@domain/1' from='requester@domain/1' id='q1'>
   <query-harness xmlns='http://ntaforum.org/2011/harness'
                  harness='http://example.org/scp'/>
</iq>

<iq type='result' from='provider@domain/1' to='requester@domain/1' id='q1'>
   <query-harness xmlns='http://ntaforum.org/2011/harness'
                  harness='http://example.org/scp'
                  xml:lang='en'>
      <label>Sawmill Control Panel</label>
      <tooltip>A harness for controlling and monitoring sawmill operations</tooltip>
      <actionDecl name='getStatus'>
         <label>Get Status</label>
         <tooltip>Fetch information about current operating status</tooltip>
         <response>
            <item name='isOperating'>
               <label>Operating</label>
               <tooltip>If true, sawmill is currently operating</tooltip>
               <datatype>boolean</datatype>
            </item>
         </response>
      </actionDecl>
      <actionDecl name='setFlowRate'>
         <label>Set Flow Rate</label>
         <tooltip>Configure the flow rate of timber into the saw</tooltip>
         <parameter name='rate'>
            <label>Rate</label>
            <tooltip>The rate to which the flow will be set</tooltip>
            <datatype>decimal</datatype>
            <units>ft/sec</units>
         </parameter>
      </actionDecl>
      <eventDecl name='shutdown'>
         <description>The sawmill line has shut down</description>
      </eventDecl>
   </query-harness>
</iq>
```

Opening an Interactive Session

If the provider indicates that it supports the "visible_and_interactive" session mode, then a requester can open a session using that mode with the intention of monitoring user activity of the tool and events that may occur as long as that session remains open.

The way in which the requester finds the provider is beyond the scope of this document. For more information, see Tool Registration and Discovery specification. The activation procedure may result in an activation reference string (see Tool Registration and Discovery section on Activating a Tool). If so, then this activation reference string is included in the open packet as shown in the example below.

Note that while not shown here, the harness may be declared to allow the open request to be parameterized as well – providing the requester with additional capability to affect the initial state and/or behavior of that session.

Example 4. Opening a new interactive session with a corresponding response

```
<iq type='set' to='provider@domain/1' from='requester@domain/1' id='o1'>
   <open xmlns='http://ntaforum.org/2011/harness'
         harness='http://example.org/scp'
         mode='visible_and_interactive'>
      <activationRef>435262</activationRef>
```



```
   </open>
</iq>

<iq type='result' from='provider@domain/1' to='requester@domain/1' id='o1'>
   <response xmlns='http://ntaforum.org/2011/harness' session='saw1'>
      <result>pass</result>
   </response>
</iq>
```

User Activity Notification

During a session opened via a harness using the visible_and_interactive mode, the provider will send notification messages to the requester about actions taken via the provider's user interface in a form consistent with the harness declaration – in other words, suitable for later being performed using the same harness in an automation mode.

In this example, the human user has clicked a button on the user interface to read the current operating mode of the sawmill, which indicates that the mill is operating.

Example 5. User activity notification sent to requester

```
<message from='provider@domain/1' to='requester@domain' id='a1'>
   <notify-action xmlns='http://ntaforum.org/2011/harness' session='saw1'>
      <started>2011-07-03T13:44:02-08:00</started>
      <action harness='http://example.org/scp'>getStatus</action>
      <responseItem name='isOperating'>true</responseItem>
      <result>pass</result>
   </notify-action>
</message>
```

Opening an Automated Session

In this use case, the requester is interested in automating the operation of the provider using the harness. Therefore, a request is made to open a session in invisible_and_automated mode. In this mode, the provider is not expected to provide any user interface. Essentially, the requester is now using a domain-specific API on that tool.

Again, note that the requester may have to perform additional work before opening a session. See Tool Registration and Discovery. The activation reference included in the open packet typically comes from the activation procedure described in that specification.

Example 6. Opening an automated session

```
<iq type='set' to='provider@domain/1' from='requester@domain/1' id='o2'>
   <open xmlns='http://ntaforum.org/2011/harness'
         harness='http://example.org/scp'
         mode='invisible_and_automated'>
      <activationRef>38472859</activationRef>
   </open>
</iq>

<iq type='result' from='provider@domain/1' to='requester@domain/1' id='o2'>
   <response xmlns='http://ntaforum.org/2011/harness' session='saw2' >
      <result>pass</result>
   </response>
</iq>
```

Performing an Action using an Automated Session

In this use case, the requester, having opened an automated session wants to perform one of the actions that the provider has declared support for in the harness declaration. Note that this request may echo a request previously

captured by the requester via an activity notification (see above). But it is equally valid that the requester may have allowed a user to describe the action sequence to be performed, without ever having used capture on an interactive session.

In this example, the requester sends a "request" packet to the provider, identifying the session on which the action is to be taken, the harness of which it is part and information about the action to be performed. (Although a simple example is shown here, the "request" element may contain any additional information that is allowed according to the harness declaration for the requested action.)

The provider responds promptly (within a few seconds) to the request with a response. (If the action will take longer to complete, the response may provide a result of "pending". See Performing a Long-Running Operation below.) It is recommended that the requester set a timeout of no less than 10 seconds to receive the IQ result packet – which should allow for nominal network delays, etc.

Example 7. Issuing a request to perform an action via the harness and the corresponding response

```
<iq type='set' to='provider@domain/1' from='requester@domain/1' id='r1'>
   <request xmlns='http://ntaforum.org/2011/harness' session='saw2'>
         <action harness='http://example.org/scp'>getStatus</action>
   </request>
</iq>

<iq type='result' from='provider@domain/1' to='requester@domain/1' id='r1'>
   <response xmlns='http://ntaforum.org/2011/harness' session='saw2'>
      <result>pass</result>
      <item name='isOperating'>false</item>
   </response>
</iq>
```

Session Resource Availability

In many cases, a provider may have a limit on the number of sessions it supports. For example, if the provider represents a real hardware resource, it would likely only support a single active session.

If a provider has reached its session limit and therefore is unable to accept new sessions despite being online, it should issue a "not available" presence notification using the standard XMPP mechanism.

If a provider does issue "not available" presence notifications, then once a session becomes available, the provider must issue an "available" presence notification.

Example 8. Presence Update

```
<presence>
   <show>xa</show>
   <status>No more sessions available</status>
   <priority>55</priority>
</presence>
```

Performing a Long-Running Operation

Some operations may take a long time to perform. In these cases, the provider must respond promptly (within a few seconds) to the request. This tells the requester that the action has been started, and that the requester will be informed later when it is completed, possibly following progress notifications. The final response is returned in a Message packet to the requester that identifies the original request ID (the ID of the XMPP packet carrying the original request) and contains a response element.

Note that opening a session is, for these purposes, a special case of a request. Returning "pending" for open tells the requester that the session is being opened, but will not be finally open until the requester receives the final response accordingly.

Example 9. A request for a long-running action, the immediate response, and the later completion notification

```
<iq type='set' to='provider@domain/1' from='requester@domain/1' id='r2'>
   <request xmlns='http://ntaforum.org/2011/harness' session='saw2'>
      <action harness='http://example.org/scp'>setFlowRate</action>
      <parameter name='rate'>41.24</parameter>
   </request>
</iq>

<iq type='result' from='provider@domain/1' to='requester@domain/1' id='r2'>
   <response xmlns='http://ntaforum.org/2011/harness' session='saw2'>
      <result>pending</result>
   </response>
</iq>

...

<message from='provider@domain/1' to='requester@domain' id='r2'>
   <response xmlns='http://ntaforum.org/2011/harness' session='saw2' requestId='r2'>
      <result>pass</result>
   </response>
</message>
```

Event Notification

While a session is underway (regardless of mode), the provider may, at any time, notify the requester of that session about certain events that occur. These events must be declared in the harness declaration. While not shown here, events may be declared to carry any kind of information – similar to the information returned in a response.

In this example, the sawmill control application has detected that the sawmill has shutdown and it fires an event accordingly to the requester that started that application's session.

Example 10. An event notification

```
<message from='provider@domain/1' to='requester@domain' id='e1'>
   <event xmlns='http://ntaforum.org/2011/harness'
        session='saw2'
        harness='http://example.org/scp'
        name='shutdown'>
      <timestamp>2011-07-03T14:01:24-08:00</timestamp>
   </event>
</message>
```

Closing a Session

When the requester is finished using the session, it can issue a close request. The close request accepts no parameters and returns nothing.

For information on deactivating the tool that is being used for the session, see the Tool Registration and Activation specification, and particularly the section on deactivation.

Example 11. A session close request and response

```
<iq type='set' to='provider@domain/1' from='requester@domain/1' id='c1'>
   <close xmlns='http://ntaforum.org/2011/harness' session='saw2'/>
</iq>

<iq type='result' from='provider@domain/1' to='requester@domain/1' id='r1'>
   <response xmlns='http://ntaforum.org/2011/harness' session='saw2'>
      <result>pass</result>
   </response>
</iq>
```


Cancelling a Request in Progress

Anytime after a request is issued, the requester may issue a cancellation for that request. If the provider has already completed the request, then the cancellation is ignored. If, however, the request is still in progress, the provider must provide a response (immediate or deferred, if a "pending" result was originally sent) with the result "abort". The provider may choose whether to include any other response data in the response or not.

Example 12. A long-running action cancelled by the requester

```
<iq type='set' to='provider@domain/1' from='requester@domain/1' id='r3'>
   <request xmlns='http://ntaforum.org/2011/harness' session='saw2'>
      <action harness='http://example.org/scp'>setFlowRate</action>
      <parameter name='rate'>25.0</parameter>
   </request>
</iq>

<iq type='result' from='provider@domain/1' to='requester@domain/1' id='r3'>
   <response xmlns='http://ntaforum.org/2011/harness' session='saw2'>
      <result>pending</result>
   </response>
</iq>

...

<message to='provider@domain/1' from='requester@domain' id='cancel1'>
   <cancel xmlns='http://ntaforum.org/2011/harness' session='saw2' requestId='r3'/>
</message>

<message from='provider@domain/1' to='requester@domain' id='r3'>
   <response xmlns='http://ntaforum.org/2011/harness' session='saw2' requestId='r3'>
      <result>abort</result>
   </response>
</message>
```

Getting Progress Updates

Especially in the case where there is a human involved on the requester side, it can be helpful to be able to report to that user about progress being made on long-running operations. For that reason, the provider must include periodic progress notifications to the requester during long-running operations. The progress notification interval should typically be between five and fifteen seconds but must be no longer than one minute. The information returned in the progress update is designed to be suitable, for example, to populate a progress dialog box. The provider may then send progress notifications periodically until the action is completed.

Example 13. A long-running action with progress updates

```
<iq type='set' to='provider@domain/1' from='requester@domain/1' id='r4'>
   <request xmlns='http://ntaforum.org/2011/harness' session='saw2'>
      <action harness='http://example.org/scp'>setFlowRate</action>
      <parameter name='rate'>34.432</parameter>
   </request>
</iq>

<iq type='result' from='provider@domain/1' to='requester@domain/1' id='r4'>
   <response xmlns='http://ntaforum.org/2011/harness' session='saw2'>
      <result>pending</result>
   </response>
</iq>

...

<message from='provider@domain/1' to='requester@domain' id='p1'>
   <progress xmlns='http://ntaforum.org/2011/harness' session='saw2' requestId='r4'>
      <totalWork>55</totalWork>
      <remainingWork>24</remainingWork>
      <status>Reconfiguring input flow motors</status>
   </progress>
```



```
</message>

<message from='provider@domain/1' to='requester@domain' id='p2'>
   <progress xmlns='http://ntaforum.org/2011/harness' session='saw2' requestId='r4'>
      <totalWork>55</totalWork>
      <remainingWork>49</remainingWork>
      <status>Restarting line after modifying flow rate</status>
   </progress>
</message>

<message from='provider@domain/1' to='requester@domain' id='n5'>
   <response xmlns='http://ntaforum.org/2011/harness' session='saw2' requestId='r4'>
      <result>pass</result>
   </response>
</message>
```

Session Close Notification

There are cases where the provider of a session may decide to close that session unilaterally (as opposed to in response to a "close" packet from the requester.) This may happen for a variety of reasons. First, if the provider has a user interface with a human user behind it, that user may decide to close the provider's UI and before terminating, it should notify the requester who started that session accordingly. But there are other reasons that a provider may choose to close a session. For example, some providers may choose to unilaterally close a session if it is holding valuable resources that have been unused by the requester for a long period of time.[1]

Regardless of the reason, the provider is required to notify the requester using a message whenever it chooses to unilaterally close a session it is providing. No notification is required when the session is closed as a result of "close" packet – as in this case, the iq packet (with type='result') returned by the server fulfills that confirmation purpose.

Example 14. A provider notifies a requester that it has closed a session that was previously opened

```
<iq type='set' to='provider@domain/1' from='requester@domain/1' id='o1'>
   <open xmlns='http://ntaforum.org/2011/harness'
         harness='http://example.org/scp'
         mode='visible and interactive'>
      <activationRef>38472859</activationRef>
   </open>
</iq>

<iq type='result' from='provider@domain/1' to='requester@domain/1' id='o1'>
   <response xmlns='http://ntaforum.org/2011/harness' session='saw1'>
      <result>pass</result>
   </response>
</iq>

...


<message from='provider@domain/1' to='requester@domain' id='close1'>
   <notify-close xmlns='http://ntaforum.org/2011/harness' session='saw1'/>
</message>
```

3. Basic Harness Declarations

Regardless of the session mode, the notion is that a session is made up of a series of actions being performed, each of which produces a response. In addition, certain events may occur during the lifetime of the session. Each of

[1] Note that most providers should be using presence information about the requester to determine immediately if requester of an active session has gone offline – in which case, the provider will normally close the session immediately. So this “idle” policy example cited is not typically needed as a protection mechanism against failed requesters. See "Reliability" section in Implementation Notes.

these, a request, a response, and an event, are communicated between requester and provider using XML elements conforming to the schemas defined by this extension. But more than that, the contents of these elements are further constrained by the contract for these as described in the response to a query-harness inquiry.

In simple cases, a request for an action may include zero, one, or many parameters. These parameters are declared in the harness declaration for that action so that the requester knows what parameters are allowed (and/or required) and the meaning of each of these.

Request Parameters

In an actual request, each parameter is passed using its name and a value. The declaration of the parameter helps to clarify its purpose and how it may be used.

Each parameter must have a unique name within its container. In addition, the parameter declaration must also provide a "label" and "tooltip" that may help a human to understand the purpose of that parameter. (See Section 7 for dealing with localization of this information.) That is all that is mandatory when declaring a parameter for an action. But the declaration may also provide many other elements to further document and/or constrain the way in which that parameter is used in an actual request. Let's look at each part of the parameter declaration. (Those shown with an * are required. Others are optional in the declaration.)

*** name**: A unique identifier for this parameter within its container that will be used to identify it in the actual request

*** label**: A short label or name for this parameter oriented towards humans – which may be locale-specific

*** tooltip**: A brief description explaining the meaning and purpose of the parameter

helpURI: A URI (typically a URL) referencing additional information on the parameter – typically hosted on a public website somewhere

mandatory: A flag (true or false) indicating whether a request must include this parameter to be valid. This defaults to true. If false, then it is recommended that a default value be provided (see below).

default: A value for this parameter that will be used if the parameter is not mandatory and is omitted from the request.

datatype: Indicates the data type appropriate for values for this parameter. This must be one of the following: string, integer, boolean, decimal, anyURI, or dateTime. These correspond to a subset of standard XSD data types and the format of the values should conform to those same standards. For example, a timestamp parameter should be formatted according to a standard that looks like "2011-07-04T14:22:52-08:00" – showing year, month, day, hour, minute, second, and local offset from GMT.

units: Typically only used for "integer" and "decimal" data types, in which case, this documents the units (if appropriate). For example, if the parameter represents a duration, you might choose to specify a datatype as "integer" with units as "milliseconds" or might choose datatype as "decimal" and units as "seconds". There is no constraint on what constitutes valid units.

masked: This is either true or false and defaults to false. If true, this means that the contents of the parameter are sensitive (such as a password) and should normally be hidden from view by users. Note that the value of the parameter is still carried unencrypted in the actual request and we depend upon encryption services, if required, on the XMPP streams itself to provide secure exchange. Any entity dealing with parameter values anywhere in the system is required to treated masked parameters with extra care for security reasons.

isMultiline: This is true or false and defaults to false. It is only applicable if the datatype is 'string' and indicates whether the string is allowed to span multiple lines. When multiline strings are used, the recommended line break is a newline character (0x0A), although recipients should be prepared to deal with other variants (such as CR or CRLF).

allowedValues: In cases where only a certain fixed set of values are appropriate for this parameter, then the allowedValues element will appear once in the parameter declaration for each of these values. Because it is

sometimes beneficial to be able to show users a more friendly description of different options for a given enumeration, this element also contains a "label" element that may be locale-specific that describes the corresponding value in human terms.

allowedLength: For string parameters, this establishes a minimum and/or maximum length for that string.

allowedCount: If provided, this determines whether multiple copies of the given parameter may appear in its container or not. For example, if the parameter is intended to carry a list of strings, then the datatype can be set to "string" and allowedCount can be specified to determine the number of items that may appear in the list. You can specify a minimum number, a maximum number, or both.

allowedPatterns: Sometimes string parameters are only valid when they conform to a certain pattern. For example, a parameter might carry a phone number, in which case the string should conform to a certain pattern. The allowedPattern provides a regular expression that must match the candidate parameter value for it to be considered valid. If you provide multiple allowedPatterns, then a match with any of these will be considered valid.

allowedRanges: For numeric parameters (int and decimal), an allowed range indicates the minimum and/or maximum values that will be considered valid. If both are specified and minimum is greater than maximum, this implies that the range between these values is invalid rather than valid. (One can think of valid range as "wrapping" around infinity.)

enablementValue: It is common that some parameters only make sense in a request depending on the presence and/or values of other parameters. If the enablementValue element is included in the declaration, then it allows the provider to indicate that this parameter is only valid if and when another certain specified parameter (by name) within the same container carries a certain value. For example, you might have another parameter called "useIndirectValues" with datatype "boolean". This parameter's value will only be relevant when useIndirectValues is set to false. So associated with this parameter would be an enablementValue that refers to "useIndirectValues" and sets a value of "false".

As you can see, the declaration of a single parameter is capable of richly describing that parameter and how it must or may be used. At the same time, for simple cases, a parameter can be as simple as a name-value pair.

Example 15. Simple parameter declaration, and corresponding request

```
<iq type='result' from='provider@domain/1' to='requester@domain/1' id='q1'>
   <query-harness xmlns='http://ntaforum.org/2011/harness'
                  harness='http://example.org/example1'>
      <label>Sample harness</label>
      <tooltip>A harness for demonstration purposes only</tooltip>
      <actionDecl name='setTitle'>
         <label>Set Title</label>
         <tooltip>Set the title for the object</tooltip>
         <parameter name='title'>
            <label>Title</lable>
            <tooltip>You can set the title to any string</tooltip>
         </parameter>
      </actionDecl>
   </query-harness>
</iq>

...

<iq type='set' to='provider@domain/1' from='requester@domain/1' id='r1'>
   <request xmlns='http://ntaforum.org/2011/harness' session='ss1'>
      <action harness='http://example.org/example1'>setTitle</action>
      <parameter name='title'>New Object Title</parameter>
   </request>
</iq>
```

For subsequent examples, for clarity, the parameter declaration is shown by itself, rather than in the context of the actual harness declaration. For each a fragment a request is shown that includes a parameter that is valid

according to the spec. (Note that the actions reflected here are not covered by any complete harness declaration in this document.)

Example 16. An optional integer parameter with a restricted range

```
<parameter name='numPeople'>
   <label># People</label>
   <tooltip>The number of people to be invited for dinner</tooltip>
   <mandatory>false</mandatory>
   <default>4</default>
   <allowedRange>
      <min>1</min>
      <max>10</max>
   </allowedRange>
</parameter>

...

<iq type='set' to='provider@domain/1' from='requester@domain/1' id='r1'>
   <request xmlns='http://ntaforum.org/2011/harness' session='ss1'>
      <action harness='http://example.org/example1'>selectPeople</action>
      <parameter name='numPeople'>5</parameter>
   </request>
</iq>
```

Example 17. An enumerated sort order parameter, enabled based on another boolean sort parameter

```
<parameter name='sort'>
   <label>Sort attendees</label>
   <tooltip>If true, attendees will be sorted</tooltip>
   <datatype>boolean</datatype>
   <mandatory>false</mandatory>
   <default>true</default>
</parameter>
<parameter name='sorting'>
   <label>Sort Order</label>
   <tooltip>The order in which people will be arranged</tooltip>
   <allowedValue label='By Age'>age</allowedValue>
   <allowedValue label='By Weight'>weight</allowedValue>
   <allowedValue label='By Height'>height</allowedValue>
   <allowedValue label='Alphabetically by Last Name'>lastName</allowedValue>
   <enablementValue>
      <parameter>sort</parameter>
      <value>true</value>
      <enableOn>equal</enableOn>
   </enablementValue>
</parameter>

...

<iq type='set' to='provider@domain/1' from='requester@domain/1' id='r1'>
   <request xmlns='http://ntaforum.org/2011/harness' session='ss1'>
      <action harness='http://example.org/example1'>organize</action>
      <parameter name='sort'>true</parameter>
      <parameter name='sorting'>weight</parameter>
   </request>
</iq>
```

Example 18. A parameter that allows a list of constrained strings to be provided

```
<parameter name='names'>
   <label>Attendee Names</label>
   <tooltip>The list of full names of attendees in the form Lastname, Firstname</tooltip>
   <allowedPattern>\S+,\S+</allowedPattern>
   <allowedCount>
      <min>1</min>
   </allowedCount>
```



```
</parameter>

...

<iq type='set' to='provider@domain/1' from='requester@domain/1' id='r1'>
   <request xmlns='http://ntaforum.org/2011/harness' session='ss1'>
      <action harness='http://example.org/example1'>chooseAttendees</action>
      <parameter name='names'>Collins,Tom</parameter>
      <parameter name='names'>Daniels,Jack</parameter>
      <parameter name='names'>Mark,Makers</parameter>
   </request>
</iq>
```

Response Items

Just as a simple request may take one or a few parameters, each response may return zero, one, or several "items" of information in return. As we will see later, the response can also carry more richly structured data, but for the simple cases, you can think of a response as consisting of a set of name-value pairs. But, just like for the request, there is a question as to how the requester should interpret this information returned for a given requested action. Hence, the harness declaration includes a "response" declaration for each action. If the action will return nothing other than a result (pass-fail) for the action, then the response declaration can be omitted. But if it is going to return any information, then it must also document what information that might be. One can think of an "item" as being the response's counterpart to a request's "parameter". Let's look at a response item declaration. (Those shown with an * are required. Others are optional in the declaration.)

*** name**: A unique identifier for this item within its container that will be used to identify it in the actual response

*** label**: A short label or name for this item oriented towards humans – which may be locale-specific

*** tooltip**: A brief description explaining the meaning and purpose of the item

helpURI: A URI (typically a URL) referencing additional information on the item – typically hosted on a public website somewhere

mandatory: A flag (true or false) indicating whether a response will always contain this item or is optional. This defaults to true. If false, then it is recommended that a default value be provided (see below).

default: A value for this item that should be inferred if the item is not mandatory and is omitted from the response

datatype: Indicates the data type appropriate for values for this item. This must be one of the following: string, int, boolean, decimal, uri, or timestamp. These correspond to a subset of standard XSD data types and the format of the values should conform to those same standards. For example, a timestamp parameter should be formatted according to a standard that looks like "2011-07-04T14:22:52-08:00" – showing year, month, day, hour, minute, second, and local offset from GMT.

units: Typically only used for "int" and "decimal" data types, in which case, this documents the units (if appropriate). For example, if the item represents a duration, you might choose to specify a datatype as "int" with units as "milliseconds" or might choose datatype as "decimal" and units as "seconds". There is no constraint on valid units.

masked: This is either true or false and defaults to false. If true, this means that the contents of the item are sensitive (such as a password) and should normally be hidden from view by users. Note that the value of the item is still carried unencrypted in the actual response and we depend upon encryption services, if required, on the XMPP streams itself to provide secure exchange. Any entity dealing with item values anywhere in the system is required to treated masked parameters with extra care for security reasons.

isMultiline: This is true or false and defaults to false. It is only applicable if the datatype is 'string' and indicates whether the string is allowed to span multiple lines. When multiline strings are used, the recommended line break

is a newline character (0x0A), although recipients should be prepared to deal with other variants (such as CR or CRLF).

allowedValues: In cases where only a certain fixed set of values are appropriate for this item, then the allowedValues element will appear once in the item declaration for each of these values. Because it is sometimes beneficial to be able to show users a more friendly description of each different value for a given enumeration, this element also contains a "label" element that may be locale-specific that describes the corresponding value in human terms.

allowedCount: If provided, this determines whether multiple copies of the given item may appear in its container or not. For example, if the item is intended to carry a list of strings, then the datatype can be set to "string" and allowedCount can be specified to determine the number of items that may appear in the list. You can specify a minimum number, a maximum number, or both.

Again, we can see that for simple responses, the declaration can be very simple. But this allows for more precise specification about what a provider may return in the response.

Example 19. Sample request and response containing a simple item

```
<iq type='result' from='provider@domain/1' to='requester@domain/1' id='q3'>
   <query-harness xmlns='http://ntaforum.org/2011/harness'
                  harness='http://example.org/example3'>
      <label>Sample harness</label>
      <tooltip>A harness for demonstration purposes only</tooltip>
      <actionDecl name='getTitle'>
         <label>Get Title</label>
         <tooltip>Get the title for the object</tooltip>
         <responseDecl>
            <item name='title'>
              <label>Title</label>
              <tooltip>The title of the object</tooltip>
            </item>
         </responseDecl>
      </actionDecl>
   </query-harness>
</iq>

...

<iq type='set' to='provider@domain/1' from='requester@domain/1' id='r3'>
   <request xmlns='http://ntaforum.org/2011/harness' session='ss2'>
      <action harness='http://example.org/example3'>getTitle</action>
   </request>
</iq>

<iq type='result' from='provider@domain/1' to='requester@domain/1' id='r3'>
   <response xmlns='http://ntaforum.org/2011/harness' session='ss2'>
      <result>pass</result>
      <item name='title'>New Object Title</item>
   </response>
</iq>
```

For subsequent examples, the item declaration is pulled out of context for clarity, and only the response is shown that demonstrates an example of that item being returned in a corresponding request.

Example 20. An optional response item that is omitted from the response

```
<item name='wasRaided'>
   <label>Party Was Raided</label>
   <tooltip>True if the party was raided by the police</tooltip>
   <datatype>boolean</datatype>
   <mandatory>false</mandatory>
   <default>false</default>
</item>

...
```



```
<iq type='result' from='provider@domain/1' to='requester@domain/1' id='r4'>
   <response xmlns='http://ntaforum.org/2011/harness' session='ss2'>
      <result>pass</result>
   </response>
</iq>
```

Example 21. A response item representing a list of enumerated values being returned

```
<item name='activities'>
   <label>Activities</label>
   <tooltip>A list of activities that took place at the party</tooltip>
   <datatype>string</datatype>
   <allowedValue label='Dancing'>dancing</allowedValue>
   <allowedValue label='Drinking'>drinking</allowedValue>
   <allowedValue label='Karaoke'>karaoke</allowedValue>
   <allowedValue label='Dinner'>dinner</allowedValue>
   <allowedValue label='Poetry Reading'>poetry</allowedValue>
   <allowedCount>
      <min>1</min>
   </allowedCount>
</items>

...

<iq type='result' from='provider@domain/1' to='requester@domain/1' id='r4'>
   <response xmlns='http://ntaforum.org/2011/harness' session='ss2'>
      <result>pass</result>
      <item name='activities'>dancing</item>
      <item name='activities'>drinking</item>
   </response>
</iq>
```

Events

During any kind of session, at any time during that session, the provider may choose to notify the requester of notable events as long as they are declared in the harness declaration returned by query-harness. A simple event can just identify the harness and the name of the event. A more complicated event may carry additional information. If an event carries additional information, this must be declared as part of the harness. An event declaration follows exactly the same model as a response. So any structure of information you can express in a response, you can equally express in an event. For that reason, the various fields describing an event item will not be repeated here – as the information listed above for an response item is identical.

Example 22. An event carrying a single timestamp item, with the corresponding event declaration shown first

```
<iq type='result' from='provider@domain/1' to='requester@domain/1' id='q4'>
   <query-harness xmlns='http://ntaforum.org/2011/harness'
                  harness='http://example.org/example4'
                  xml:lang='en'>
      <label>Sample harness</label>
      <tooltip>A harness for demonstration purposes only</tooltip>
      <eventDecl name='raided'>
         <description>The party has been raided</description>
         <item name='arrestCount'>
            <label># Arrested</label>
            <tooltip>The number of people arrested</tooltip>
            <datatype>integer</datatype>
         </item>
      </eventDecl>
   </query-harness>
</iq>

...

<message from='provider@domain/1' to='requester@domain' id='e4'>
```



```
   <event xmlns='http://ntaforum.org/2011/harness'
        session='ss4'
        harness='http://example.org/example4'
        name='raided'>
      <timestamp>2011-07-05T03:01:45-08:00</timestamp>
      <item name='arrestCount'>21</item>
   </event>
</message>
```

4. Session Modes

A provider of a given harness is allowed to declare what modes it supports for sessions using that harness. These modes are described in detail below. The provider of a harness is required to support at least one mode for its sessions, but it is free to support as many modes as it is able. Not all providers of the same harness may support the same modes -- and that is why the supported modes are not specified in the harness declaration itself, but, rather, in the list-harnesses response.

For automation-enabled tools that are essentially exposing an API, most implementers will find that mode 1 ("invisible_and_automated") is the easiest to support. And they are, of course, allowed to stop there and not bother to implement any other mode. By doing this, they are allowing other tools to consume their services. But as you will read below, there is great value in being able to support other modes as well. In fact, one might argue that if we support only mode 1 on most tools, then we will not be doing too much more than creating a higher-level set of APIs on tools that probably already support some kind of API. Perhaps these harness declarations are nicer in some ways than, say, Tcl APIs, but they are not likely to fundamentally change the test tool integration challenges. The addition of other modes enables a whole new range of possibilities -- such as capture-replay scenarios where users get to use the best of all worlds: the optimal user interfaces for each tool involved in a process, and their favorite test authoring tool, for example. It also enables things like self-documenting processes -- where information about a whole sequence of operations spanning many different tools can be automatically recorded into an auditing system, for example.

For this reason, NTAF implementers are strongly encouraged to support as many modes in their tools as possible. Note that another possibility is to create companion tools – where one tool is primarily designed for interactive use (and only exposes mode 2) while its companion is optimized for automation (and supports only mode 1).

Mode 1: "invisible and automated"

In this mode, the session provider is essentially just providing a traditional application programming interface (API) for the session. There is no human involved and no user interface on the provider side. The client requests a "session" and then invokes a set of requests for actions on the session. Essentially this is like any traditional mechanism for one computer process to talk to another computer process -- making requests to perform actions. In this mode, it works very similarly to, say, web services, or RPC mechanisms such as DCOM or RMI.

Mode 2: "visible and interactive"

This mode is for a very different situation. In this case, there is typically a human involved on the provider side. One tool asks another tool to open a session and (depending on the declaration of the "open" action in the harness declaration) may describe certain properties about the behavior of that session. But unlike mode 1 above, the intention of this session is for a human user to interact with it from that point. And, very importantly, in this mode, the provider agrees to send notifications to the originator for each action performed interactively (by the human user).

One might ask why a session would be started in this way, rather than simply having the user start a session directly using the provider tool. There are a few reasons. First, the originating tool may have information about how and when that session should be started that the human may not readily have available. For example, the originating tool may start the session in response to some event. As a more specific example, when a failure is

discovered during testing, the testing tool may decide that a bug report is appropriate. In that case, it could request a session from a bug tracking tool, and could provide the initial information to populate that bug report -- allowing the user to do additional work in that session, perhaps finalizing and filing the bug report.

A second reason for this mode is for documenting a process. If a tool supports this second mode for a given harness, then it is also agreeing to notify the originating tool about actions performed by the human user during the lifetime of that session. And these notifications are in the form of requests and responses for actions declared in the harness itself. This is extremely valuable in certain scenarios, because the originating tool can use this information to document processes performed by humans. One good example of where this is valuable is in a capture-replay system, where the human actions are being used to prototype steps in an automated test case. Note, also, that an authoring tool (the originator in this case) may be documenting many concurrent sessions when dealing with system testing, for example.

Example 23. A session opened for interactive use, and a corresponding set of user activity notifications issued from the provider back to the requester

```
<iq type='set' to='provider@domain/1' from='requester@domain/1' id='o2'>
   <open xmlns='http://ntaforum.org/2011/harness'
         harness='http://example.org/scp'
         mode='visible_and_interactive'>
      <activationRef>38472859</activationRef>
   </open>
</iq>

<iq type='result' from='provider@domain/1' to='requester@domain/1' id='o2'>
   <response xmlns='http://ntaforum.org/2011/harness' session='saw2'>
      <result>pass</result>
   </response>
</iq>

...

<message from='provider@domain/1' to='requester@domain' id='c1'>
   <notify-action xmlns='http://ntaforum.org/2011/harness' session='saw2'>
      <started>2011-07-04T14:02:24-08:00</started>
      <action harness='http://example.org/scp'>setFlowRate</action>
      <requestParameter name='rate'>50.0</requestParameter>
      <result>pass</result>
      <duration>31.42</duration>
   </notify-action>
</message>

...

<message from='provider@domain/1' to='requester@domain' id='c2'>
   <notify-action xmlns='http://ntaforum.org/2011/harness' session='saw2'>
      <started>2011-07-04T14:03:11-08:00</started>
      <action harness='http://example.org/scp'>getStatus</action >
      <result>pass</result>
      <responseItem name='isOperating'>true</responseItem>
   </notify-action>
</message>

...

<message from='provider@domain/1' to='requester@domain' id='c2'>
   <notify-close xmlns='http://ntaforum.org/2011/harness' session='saw2'/>
</message>
```

If the requester has no interest in the captured user activity, it may indicate this to the provider via a "requestUserActivity" attribute in the open request – passing "false" if it doesn't wish to receive user activity notifications.

Mode 3: "visible and automated"

This third mode requires the provider to present a user interface suitable for humans to view during the lifetime of that session, and yet the user interface is typically read-only -- because the actions performed on that session will still be driven by the originator as a sequence of actions requested using the harness protocol. As these actions are performed, the user interface presents the appropriate information to the human user -- much as if that user had performed those same (or equivalent) actions.

Again, one might wonder as to the value of such a mode. The answer is that mode 1 will normally be used in a fully automated process. But while developing that automated process, it is often very valuable for debugging to be able to see the process as it operates. Also, when there are problems in an automated process the ability to troubleshoot problems is greatly enhanced if a user can access user interfaces for each of the sessions involved in that process.

Mode Switching

There is a fourth scenario that is not directly addressed by these modes. This is the case where an automated process is running in mode 3 but at some point during that process it becomes valuable to be able to switch between mode 3 and mode 2. For example, suppose that the originating tool is a test execution tool. It is running a script that is, among other things, driving an automated session that it started in mode 3. If that execution tool supports breakpoints for debugging, it can be extremely valuable if it is also able to make the user interfaces for the sessions interactive at that point. That empowers a test developer, for example, to interact with any of those sessions that have a user interface, allowing them to quickly find the source of certain problems and to probe certain data. This fourth scenario is not directly addressed as its own mode. Instead, this is really a capability of the provider to switch between modes 2 and 3. The mechanism for switching between modes could be a harness specifically for this purpose, but is beyond the scope of this extension.

5. Advanced Features

XML Parameters

There are occasions where part or all of the information required for a given action on a harness may need to conform to some formal XML document specification – typically used and defined in a more general context. In this case, it would not make sense to have to reproduce that information as various individual parameters in the declaration of the action.

In some sense, this is a case where the information to be provided is effectively a file that conforms to a certain XML schema. And, while using an attachment (see below) might be one way to accomplish this, there are reasons why it may be preferable to include that information directly inside the request itself.

For this reason, as part of a harness declaration, an action may indicate that in additional to normal parameters, it may also indicate that it must or may accept so-called "xml parameters". Instead of the value appearing in a text field in a parameter tag of the request, it will appear as an XML document fragment inside an xml parameter tag.

In this case, the harness declaration identifies an element name and a specific XML namespace in which that element must appear. In this way, the contract for information exchange remains rigorous between client and server (since the namespace plus element implies a specific schema definition).

Note the parallel between this mechanism and IO Data (XEP-0244) where each request accepts exactly one input which is an XML document fragment and produces one output, which is another XML document fragment. This harness mechanism effectively devolves into equivalence with IO Data when each action is declared with a single xml parameter.

Example 24. An XML parameter declaration, and an example of a request using that declaration

```
<actionDecl name='setConfiguration'>
```



```
   <label>Set Configuration</label>
   <tooltip>Set up the configuration using an XML</tooltip>
   <xmlParameter name='config'>
      <label>Configuration</label>
      <element>device-configuration</element>
      <xmlNamespace>http://example.org/schemas/sawmill/configuration/1.0</xmlNamespace>
   </xmlParameter>
</actionDecl>

...

<iq type='set' to='provider@domain/1' from='requester@domain/1' id='x1'>
   <request xmlns='http://ntaforum.org/2011/harness' session='saw2'>
       <action>setConfiguration</action>
       <xmlParameter name='config'>
          <device-configuration xmlns='http://example.org/schemas/sawmill/configuration/1.0'>
             <flowRate>24.252</flowRate>
             <boardWidth>wide</boardWidth>
             <timberType>softwood</timberType>
          </device-configuration>
       </xmlParameter>
   </request>
</iq>
```

In this example, the harness will accept a set of configuration information that can be expressed in XML that conforms to some specific schema that is explicitly declared in the action declaration. Even more specifically, the declaration indicates the root element name from that schema that must appear when a request is made. In this way, the contract is specific on the structure of the content that will be delivered.

XML Response Items

Just as there are occasions on which passing a formal XML document fragment is valuable, it is also the case sometimes where returning an XML document fragment is useful. To mean this need, there is a parallel for responses called XML response items. Similar to parameters, in such a case, the harness declaration describes the element name and namespace for a document to be returned. And the response contains that item by encapsulating that element inside the appropriate item.

Example 25. An XML response item declaration, and a response that conforms to it

```
<actionDecl name='getContract'>
   <label>Get Contract</label>
   <tooltip>Fetch the current operating contract</tooltip>
   <xmlItem name='contract'>
      <label>Contract</label>
      <element>contract</element>
      <xmlNamespace>http://example.org/schemas/timber/contract</xmlNamespace>
   </xmlItem>
</actionDecl>

...

<iq type='set' to='provider@domain/1' from='requester@domain/1' id='x1'>
   <request xmlns='http://ntaforum.org/2011/harness' session='saw2'>
       <action>getContract</action>
   </request>
</iq>

<iq type='result' from='provider@domain/1' to='requester@domain/1' id='x1'>
   <response xmlns='http://ntaforum.org/2011/harness' session='saw2'>
      <result>pass</result>
      <xmlItem name='contract'>
         <contract xmlns='http://example.org/schemas/timber/contract'>
            <contractId>242-52969-22</contractId>
            <signed>2011-04-01</signed>
            <value currency='usd'>11425306</value>
         </contract>
```



```
      </xmlItem>
   </response>
</iq>
```

Request Attachments

Occasionally, there may be cases where an action to be performed will require certain files in order to perform that action. These files may not be XML files and therefore are ineligible to be carried using XML parameters as described above. XMPP provides an extension for supporting file transfers (XEP-0096) and an extension using Jingle (XEP-0234). **[The selection of which to use is TBD.]**

To use attachments, the action declaration may include any number of "files" elements that are used to describe the files that may or must be "attached" to the request. The declaration may specify what file extensions are appropriate for a given attachment (which the expectation that a human user may be making the actual file selection from their own machine and this helps them narrow that selection appropriately). The declaration also allows for one or many files to be attached for a given named file declaration on a given action. In this way, for example, the client, if allowed, may transfer several files for a given named file in the declaration.

As part of the actual XMPP request packet, the client will describe the file(s) it is going to transfer to the provider. As soon as the request is sent, the client will then initiate file transfers using XMPP file transfer for each of these using the file names that it provided in the request. The provider will proceed with the action as soon as it receives all of the files indicated.

Example 26. An action declaration that includes a file specification, and a corresponding request

```
<actionDecl name='addProfiles'>
   <label>Add Profiles</label>
   <tooltip>Add one or more operating profile documents into the working set</tooltip>
   <file name='profile'>
      <label>Operating Profile</label>
      <allowedCount>
         <min>1</min>
         <max>10</max>
      </allowedCount>
      <allowedFileExtension>pro</allowedFileExtension>
      <allowedFileExtension>prox</allowedFileExtension>
   </file>
</actionDecl>

...

<iq type='set' to='provider@domain/1' from='requester@domain/1' id='f1'>
   <request xmlns='http://ntaforum.org/2011/harness' session='saw2'>
       <action>addProfile</action>
       <file name='profile'>
          <filename>primary.pro</filename>
       </file>
       <file name='profile'>
          <filename>secondary.prox</filename>
       </file>
   </request>
</iq>

(XMPP file transfer from requester@domain/1 to provider@domain/1 for file primary.pro...)

(XMPP file transfer from requester@domain/1 to provider@domain/1 for file secondary.prox...)

<iq type='result' from='provider@domain/1' to='requester@domain/1' id='f1'>
   <response xmlns='http://ntaforum.org/2011/harness' session='saw2'>
      <result>pass</result>
   </response>
</iq>
```


Response Attachments

Similarly, there are cases where the provider may produce or access files that may be useful to the requester. In this case, the response declaration will include file declarations describing files that will be returned as part of the response. The response itself will include specific file information and as soon as the response has been sent, the provider will initiate file transfer procedures for those files using XMPP file transfer.

Note that for deferred responses, associated files cannot be transferred until after the final response is issued to the requester.

Example 27. A response declaration that includes a file specification, and a corresponding response.

```
<actionDecl name='getProfiles'>
   <label>Get Profiles</label>
   <tooltip>Fetch operating profile documents in the working set</tooltip>
   <fileItem name='profile'>
      <label>Operating Profile</label>
      <allowedCount>
         <min>1</min>
      </allowedCount>
   </fileItem>
</actionDecl>

...

<iq type='set' to='provider@domain/1' from='requester@domain/1' id='f2'>
   <request xmlns='http://ntaforum.org/2011/harness' session='saw2'>
       <action>getProfiles</action>
   </request>
</iq>

<iq type='result' from='provider@domain/1' to='requester@domain/1' id='f2'>
   <response xmlns='http://ntaforum.org/2011/harness' session='saw2'>
      <result>pass</result>
      <file name='profile'>
         <filename>primary.pro</filename>
      </file>
      <file name='profile'>
         <filename>secondary.prox</filename>
      </file>
   </response>
</iq>

(XMPP file transfer from provider@domain/1 to requester@domain/1 for file primary.pro...)

(XMPP file transfer from provider@domain/1 to requester@domain/1 for file secondary.prox...)
```

Request Groups

Up to this point, a request can carry any number of parameters, some of which may be allowed to repeat. This handles a large fraction of the simple cases well. However, there are two cases that are not handled well. First, there is the case where the action may carry a large set of parameters and, especially when a human is involved, there is no obvious organization to these parameters. By providing for groups of parameters, the meaning to a human user can be enhanced. Second, while an individual parameter may be declared so that it can be repeated, there is no way to easily handle cases where the action needs to be able to accept multiple instances of a more complex object. For example, the action might allow the requester to specify a list of contacts to be processed, where each contact has several properties.

To handle these needs, we introduce the idea of a "request group". This is like a complex parameter that instead of having a single value is made up of a set of parameters. The group has properties in its declaration to describe the group itself, and how it may or must be used in the request. A group can contain any number of parameter,

but may also contain any number of other groups, as well. In this way, one can create a whole hierarchy of parameters, repeating at any level.

There are cases where a repeating group is really representing rows in a conceptual table. In this case, it may be important to understand what the "key" parameter is for columns in that table. If so, the declaration can indicate this, so that the requester knows that a certain parameter in that group must have a unique value in each instance of that group sent in a request.

Example 28. An action declaration that includes a request group, and a corresponding request

```
<actionDecl name='setPermissions'>
   <label>Set Operator Permissions</label>
   <group name='permissionRecord'>
      <label>Permission Record</label>
      <allowedCount>
          <min>0</min>
      </allowedCount>
      <parameterKeyName>userid</parameterKeyName>
      <parameter name='userid'>
         <label>User ID</label>
      </parameter>
      <parameter name='privilege'>
         <label>Privilege Level</label>
         <allowedValue label='Read only'>read-only</allowedValue>
         <allowedValue label='Normal'>normal</allowedValue>
         <allowedValue label='Administrator'>admin</allowedValue>
      </parameter>
   </group>
</actionDecl>

...

<iq type='set' to='provider@domain/1' from='requester@domain/1' id='r11'>
   <request xmlns='http://ntaforum.org/2011/harness' session='saw2'>
       <action>setPermissions</action>
       <group name='permissionRecord'>
          <parameter name='userid'>jschmoe</parameter>
          <parameter name='privilege'>normal</parameter>
       </group>
       <group name='permissionRecord'>
          <parameter name='userid'>jdoe</parameter>
          <parameter name='privilege'>admin</parameter>
       </group>
       <group name='permissionRecord'>
          <parameter name='userid'>bschwedchuk</parameter>
          <parameter name='privilege'>normal</parameter>
       </group>
   </request>
</iq>
```

Response Groups

Response groups are the counterpart to request groups, except for the response. This handles cases where the provider will return information in a response organized into groups. And because response groups can indicate that they are allowed to appear multiple times in a response, this becomes a mechanism by which the provider can effectively return tables of information. And, again, this is where the notion of a "key column" may be important to the requester (e.g., to signal to the requester how best to access information for a given "cell" in the table based on the value of the cell in the same row in the key column).

Example 29. A response declaration that includes a response group representing a table, and a corresponding response

```
<responseDecl>
   <group name='log'>
      <label>Saw Logs</label>
```



```
      <allowedCount>
          <min>0</min>
      </allowedCount>
      <itemKeyName>timestamp</itemKeyName>
      <item name='timestamp'>
         <label>Timestamp</label>
         <datatype>dateTime</datatype>
      </item>
      <item name='diameter'>
         <label>Diameter</label>
         <datatype>decimal</datatype>
         <units>inches</units>
      </item>
      <item name='length'>
         <label>Length</label>
         <datatype>decimal</datatype>
         <units>feet</units>
      </item>
   </group>
</responseDecl>

...

<iq type='set' to='provider@domain/1' from='requester@domain/1' id='r12'>
   <request xmlns='http://ntaforum.org/2011/harness' session='saw2'>
       <action>getLogTable</action>
   </request>
</iq>

<iq type='result' from='provider@domain/1' to='requester@domain/1' id='r12'>
   <response xmlns='http://ntaforum.org/2011/harness' session='saw2'>
      <result>pass</result>
      <group name='log'>
         <item name='timestamp'>2011-07-04T15:39:01</item>
         <item name='diameter'>14.24</item>
         <item name='length'>41.5</item>
      </group>
      <group name='log'>
         <item name='timestamp'>2011-07-04T15:43:19</timestamp>
         <item name='diameter'>13.51</item>
         <item name='length'>61.3</item>
      </group>
      <group name='log'>
         <item name='timestamp'>2011-07-04T15:45:33</timestamp>
         <item name='diameter'>12.97</item>
         <item name='length'>50.4</item>
      </group>
   </response>
</iq>
```

Context

In some cases, a provider that is performing actions requested via a harness will need to consume services provided by yet another tool using another harness. And this chain may continue where that third tool, in turn, depends on yet another tool. There are cases where the linkage of these chains may be important to be able to understand. For example, when troubleshooting a complex process that is automated using a network of tools interacting via harnesses, it will be beneficial to be able to look at any given request and to be able to see where that request came from – all the way back up the chain of requesters. And there are other special cases where certain proprietary information about an originator "upstream" in such a chain of requests may be important to tools further downstream.

To that end, each request issued on a harness has an optional "context" element. According to the request schema, this context element must identify an entity (using that entity's Jabber ID), and a session ID representing the session that entity is operating on (if any). In addition, the context element may include any arbitrary XML

context information that the entity believes may be useful downstream. And, most importantly, the context element may include a nested context element (following the same schema) that provides the context that it received in a request (if any) that it is processing.

Here are the rules for context elements in requests:

- If an entity is issuing a request on a harness but this is not as a result of performing an action as a result of a request made on one of its own harnesses, then it is not required to include a context element in the request it makes downstream.
- In this case, the entity *may* include a context element, in which case it *should not* identify itself in the "entity" attribute (since this is redundant to the receiver with the addressing information carried in the XMPP envelope). The reason for including a context element is if this entity wishes to provide any ancillary information that may be useful for troubleshooting or as context information to other entities downstream.
- If an entity is issuing a request on a harness as part of performing an action in response to a request on one of its own harnesses, then it *must* include a context element in that request. This context element must not identify itself in the outer context element (as, again, that would be redundant to the recipient) but must identify which of its harnesses it is providing that caused it to issue this request, the session of that harness, and the request ID of the request it is processing.
- In addition, in these cases, the context element *must* include an inner context element that exactly matches the context element it received in the request that it is processing – except that it must also populate the "entity" attribute in that inner context based on the XMPP addressing in the request being processed. (This need for the provider to populate this rather than the requester upstream is to avoid any opportunity for "spoofing" that may eventually lead to security issues later.)

Example 30. A chain of causal requests through a set of providers, each carrying context information

```
<iq type='set' from='bill@domain/1' to='bob@domain/1' id='rq1'>
   <request xmlns='http://ntaforum.org/2011/harness' session='bob1'>
       <action>reset</action>
   </request>
</iq>

<iq type='set' from='bob@domain/1' to='joe@domain/1' id='rq2'>
   <request xmlns='http://ntaforum.org/2011/harness' session='joe1'>
       <action>resetDevice</action>
       <context harness='http://example.org/bobcontrol' session='bob1'>
          <details>
             <bob-context xmlns='http://example.org/bobcontext'>
                <software>BobSoft</software>
                <version>2.1.1.5</version>
             </bob-context>
          </details>
          <context entity='bill@domain/1' requestId='rq1'/>
       </context>
   </request>
</iq>

<iq type='set' from='joe@domain/1' to='jerry@domain/1' id='rq3'>
   <request xmlns='http://ntaforum.org/2011/harness' session='jerry1'>
       <action>resetPort</action>
       <context harness='http://example.org/joedevice' session='joe1'>
          <context entity='bob@domain/1'
                   harness='http://example.org/bobcontrol'
                   session='bob1'
                   requested='rq2'>
             <details>
                <bob-context xmlns='http://example.org/bobcontext'>
                   <software>BobSoft</software>
                   <version>2.1.1.5</version>
                </bob-context>
             </details>
             <context entity='bill@domain/1' requestId='rq1'/>
          </context>
```



```
        </context>
    </request>
</iq>
```

In this example, the initial request from Bill to Bob does not carry any context, because Bill is not performing the request as part of handling a request on a harness session it is providing. (Bill could, however, still choose to include a context element if he wants to communicate some kind of context information downstream.)

As a result of processing the request from Bill, Bob decides to issue a "resetDevice" request on a session that he has open to Joe. He includes a "context" tag in the request identifying the harness he is providing that required him to perform this "resetDevice" request (not to be confused with the harness he is using on Joe to do this resetDevice action). He also identifies his name for that session he, Bob, is providing to Bill. Inside that context tag, he decides to include some proprietary details about his situation that may be of interest downstream. In this case, he fills in a "bob-context" element with his version information. (Note that the use of XML namespaces is not required but is strongly encouraged in the contents of these details tags.) Bob also includes inside his context element a nested context element identifying his client – Bill in this case. Notice that Bill had not included a context element in this example. If he had, then Bob would have put that same context element in here. But even if he did receive a context element, he must fill in the originator of the request, because he knows who sent the request.

In the final packet shown in the example, Joe decides that in order to fulfill his requested action to resetDevice, he must, in turn, invoke a "resetPort" action using a session that he has open to Jerry. Again, he includes a context element in the request identifying the harness and session that he, Joe, is working on. Inside that context element, he has no particular details of his own to pass along, so he just nests the context element that he received in his request. The one thing he does, however, is to update the inner top-most context element to set the entity attribute based on who issued the request to him – bob@domain/1 in this example.

At this point, Jerry, who receives the resetPort request will have some interesting information about where this request came from. He will be able to see from the context structure that it all started with Bill who issued a request to Bob. He can see that Bob is running version 2.1.1.5 of BobSoft, in case that is of any interest. And he can see that Bob invoked the action on Joe because of a request from Bill.

6. Nested Harnesses

It is often the case that multiple tools may be able to support a common subset of capabilities. On the other hand, these tools are not identical, and therefore each also has functionality that is not in common with others. To handle these cases, each tool could declare multiple harnesses – one of which is the same for all tools. The problem with this is that requests are made in the context of a session. And that context may be very important. Therefore having two distinct sessions in order to support, for example, value-added commands is not optimal as it then creates a problem in identifying linkages between these sessions.

To address this challenge, this specification allows for the possibility of nested harnesses. Each harness, as part of its own declaration can identify any number of other harnesses that it can also support within the same session.

Harness authors must be careful to prevent nesting loops where harness A refers to harness B which refers to harness A. Implementers should ensure that such loops are rejected or ignored.

Suppose, as an example, one defined a harness called "http://example.org/harnesses/addressing" consisting of two actions, "setAddress" and "getAddress", and one event "addressChanged". Then any tool which may incorporate a postal address could declare in its harness that is supports this 'addressing' harness. Any consumer of these harnesses will then be able to treat all of these tools in the same way as far as addressing goes – even though they may need to treat them differently for other tool-specific functions.

Any tool that declares support for a harness is required to respond to a query-harness request with the information about that harness. This must include a reference to all nested harnesses that are included in that harness.

Example 31. A simple harness query and response that only includes a reference to a nested harness

```
<iq type='get' to='provider@domain/1' from='requester@domain/1' id='q6'>
   <query-harness xmlns='http://ntaforum.org/2011/harness'
                  harness='http://example.org/h6'>
</iq>

<iq type='result' from='provider@domain/1' to='requester@domain/1' id='q6'>
   <query-harness xmlns='http://ntaforum.org/2011/harness'
                  harness='http://example.org/h6'
                  xml:lang='en'>
      <label>Mail Labeling</label>
      <tooltip>A harness for labeling of mail</tooltip>
      <subharness>http://example.org/harnesses/addressing</subharness>
   </query-harness>
</iq>
```

As you can see in this example, the harness indicates that it supports a nested harness identified using a namespace declaration, "http://example.org/harnesses/addressing". This nested harness namespace references a unique set of action and event declarations. But this doesn't tell the requester what is possible using that nested harness. To get more details about a nested harness, the requester must issue a query-harness inquiry for that harness.

Example 32. A query for details on a nested harness

```
<iq type='get' to='provider@domain/1' from='requester@domain/1' id='q7'>
   <query-harness xmlns='http://ntaforum.org/2011/harness'
                  harness='http://example.org/harnesses/addressing'>
</iq>

<iq type='result' from='provider@domain/1' to='requester@domain/1' id='q7'>
   <query-harness xmlns='http://ntaforum.org/2011/harness'
                  harness='http://example.org/harnesses/addressing'
                  xml:lang='en'>
      <label>Postal Addressing</label>
      <tooltip>A set of actions and events related to mail addressing</tooltip>
      <actionDecl name='getAddress'>
         <label>Get Address</label>
         <tooltip>Fetch the current mailing address</tooltip>
         <responseDecl>
            <item name='streetAddress'>
               <label>Street Address</label>
               <tooltip>A list of address lines providing the street address</tooltip>
               <allowedCount>
                  <min>1</min>
               </allowedCount>
            </item>
            <item name='city'>
               <label>City</label>
               <tooltip>City name</tooltip>
            </item>
            <item name='state'>
               <label>State</label>
               <tooltip>State</tooltip>
            </item>
            <item name='postalCode'>
               <label>Postal Code</label>
               <tooltip>5 or 9 digit postal code</tooltip>
               <allowedPattern>[0-9]{5}(\-[0-9]{4})?</allowedPattern>
            </item>
         </response>
      </actionDecl>
      <actionDecl name='setAddress'>
```



```
         <label>Set Address</label>
         <tooltip>Update the current mailing address</tooltip>
         <parameter name='streetAddress'>
            <label>Street</label>
            <tooltip>One or more lines of street address</tooltip>
               <allowedCount>
                  <min>1</min>
               </allowedCount>
         </parameter>
         <parameter name='city'>
            <label>City</label>
            <tooltip>City name</tooltip>
         </parameter>
         <parameter name='state'>
            <label>State</label>
            <tooltip>State name</tooltip>
         </parameter>
         <parameter name='postalCode'>
            <label>Postal Code</label>
            <tooltip>5 or 9 digit postal code</tooltip>
            <allowedPattern>[0-9]{5}(\-[0-9]{4})?</allowedPattern>
         </parameter>
      </actionDecl>
      <eventDecl name='addressChanged'>
         <description>The current address has changed</description>
      </eventDecl>
   </query-harness>
</iq>
```

During automated sessions (and when notifying using notify-action) requests using actions that are defined in nested harnesses are made by fully qualifying the action name requested using the name for that nested harness from the harness declaration.

Example 33. A request using an action defined in an a nested harness

```
<iq type='set' to='provider@domain/1' from='requester@domain/1' id='r10'>
   <request xmlns='http://ntaforum.org/2011/harness' session='post1'>
      <action harness='http://example.org/harnesses/addressing'>getAddress</action>
   </request>
</iq>

<iq type='result' from='provider@domain/1' to='requester@domain/1' id='r10'>
   <response xmlns='http://ntaforum.org/2011/harness' session='post1'>
      <result>pass</result>
      <item name='streetAddress'>515 Maple St.</item>
      <item name='streetAddress'>Suite 5100</item>
      <item name='city'>Centerville</item>
      <item name='state'>Kansas</item>
      <item name='postalCode'>51105-3311</item>
   </response>
</iq>
```

Example 34. An event notification for an event defined in a nested harness

```
<message from='provider@domain/1' to='requester@domain' id='e4'>
   <event xmlns='http://ntaforum.org/2011/harness'
        session='ss4'
        name='addressChanged'
        harness='http://example.org/harnesses/addressing'>
      <timestamp>2011-07-05T09:22:31-08:00</timestamp>
   </event>
</message>
```


7. Localization

Since humans may be involved in tools on either or both sides of these exchanges, we must deal with the fact that the harness provider may need to adapt to the locale of the requester. For example, if a native Mandarin speaker will be using the tool that will configure properties about actions to be performed on another tool, that user would prefer to have labels, tooltips, help text, etc., displayed in Mandarin.

To accommodate this, the specification allows certain elements to use the standard <xml:lang> mechanism specified in Section 2.12 of the XML standard of the W3C. This element is provided in requests to indicate what language the recipient would *prefer* to receive its responses in. Likewise, this element is provided in responses to indicate what language the content is actually delivered in. This specification does not require that the provider respond using the requested language, but best efforts are encouraged. In any case, when providing a query-harness response, the provider must identify the language, even if it does not match the one requested.

8. Harness Compatibility and Evolution

It may be very common for a large number of tools to support the same harness. This specification requires that all implementers of a given harness will respond with an identical declaration of that harness in response to a query-harness. (An exception to this is made for providing locale-specific user information which does not affect the fundamentals of the harness contract, but only the user-visible strings.) Because of this fact, clients need never fetch information about a given harness more than once and this greatly improves the scalability and performance of the overall solution – because clients can be assured that if they encounter a tool that supports a harness with a certain name, that all other tools supporting that harness will provide an identical service.

The same is true for nested harnesses. All providers of a nested harness must respond identically (except perhaps for locale reasons) to a query-harness to that nested harness.

With this in mind, one must pay careful attention to the evolution of a given harness. Once "published", a harness may never change. Even if modifications that are viewed as fully backwards compatible are made to a harness, such a change is still disallowed. However, a mechanism is provided that addresses this need for evolution while not imposing a large burden on harness implementers. In each harness declaration, there is an optional field for specifying another harness that this harness is intended to supercede. The new version of a harness must have a new unique name (usually with some suffix providing a version of some kind). The client, however, must not infer any information from the naming of these various harnesses. Instead, when a client encounters a harness that indicates that it supercedes another harness, that client will know that this is the preferred harness to use. However, the client may choose to use the earlier harness if it is still supported by the provider in cases where, for example, it has already built some asset that depends on the earlier harness.

One might worry that having to support two harnesses when the new harness is only a slight extension to the earlier version of that harness will be a large burden. However, from the provider standpoint, this should not be the case. As long as the implementation is done carefully, a single implementation of the harness should be able to handle both new and old harness declarations. For example, if a newer version of a harness adds support for a new action that was not declared on the old harness, then the provider can use the same harness implementation to deal with requests for session using the new or old harness – because users of the older harness will simply not make requests for the new action. Likewise, if a parameter is added to an existing action, the provider will normally make that parameter optional with a reasonable default value, so that requests using the older harness will work because the default value will be used. And if there are changes (rather than extensions) to the harness, then the implementation can add a few isolated checks to see which version of the harness is being used on the session to know how to respond.

9. Error Handling

Error handling for this extension should conform to the XMPP Core specifications. However, one should note the distinction between an error involving the service itself versus errors encountered while performing a requested action. The "result" element returned with each response in a session provides a means of reporting cases where the requested action failed or was aborted for some reason. In these cases, one would not use XMPP error reporting mechanisms but would, instead, report these using the response itself. For all other errors, standard XMPP error reporting mechanisms should be used.

For example, if a provider receives a request to open a session using a harness it does not support, then it should return an iq packet with type = 'error' and with an error code <feature-not-implemented>. Likewise, if it receives a request for a session that it does not recognize or has been closed, it should return an <item-not-found>. For more information, see XEP-0086.

10. Implementation Notes

Reliability

Implementers of all NTAF-enabled tools should consider reliability carefully. XMPP is carried over TCP and therefore information flows are reliable. However, this can lead to a false sense of security. A problem occurs if the provider or requester involved in a harness session goes offline while the session is underway or, worse, when a long-running request is being processed by the provider. Since the TCP sockets are between the XMPP clients and their server (rather than end-to-end between clients) there is no guarantee that a packet sent by a requester to a provider will, in fact, arrive at that provider. The inverse (provider to requester) is also true.

The solution to this problem comes from using XMPP presence information. Before a requester opens a session to a provider, it should subscribe to presence information for that provider. And before the provider accepts the requested session, it should subscribe to presence information for that requester. If either end learns that the other has gone offline, then it can close its session accordingly.

Let's look at how this ensures reliability. If a requester has opened a session and then goes offline, leaving the session open, the provider will get a presence notification and can release the resources associated with that session. If a requester has initiated a request, the provider will know that it can cancel the request in progress, since there will be no one to send the response to. If a provider has accepted a session, but later goes offline, the requester will find out that the session is no longer valid without having to depend on a timeout on a request. And if the provider goes offline while the requester is waiting for a deferred response from a long-running request, it will know by virtue of the presence information that there is no reason to wait for that response.

Without these presence subscriptions, both sides are vulnerable to dangling sessions. The requester is most vulnerable to the case where the provider goes offline while processing a long-running response, since otherwise it has no guaranteed way to know what an appropriate timeout is for a given long-running request. The provider is vulnerable at all times without presence information from the requester – as it may otherwise hold resources indefinitely waiting for the requester to issue a request or close the session.

Scalability

It may be common to encounter hundreds or more of the same kind of tool in an environment. These identical tools will respond identically to queries made to them. If a client requests harness information about the same harness from every new tool that it encounters, then it is possible that the network will be burdened with a lot of useless traffic. Moreover, the time to get this response and to process it is likely to reduce the performance of these client tools.

Therefore, it is recommended that client tools maintain a registry or cache of harness information where the key is based on the name of each item. In that way, when a tool discovers a new provider, it need only fetch harness

information for those that it does not already recognize. Note that a tool may encounter a harness it doesn't recognize, but that harness may reference a harness that it already knows about. Again, it should avoid fetching new information for the recognized harness.

During sessions, it may be tempting to request frequent progress updates on long-running operations. This temptation should be avoided as it could tend to flood the network with a lot of messages. The recommended progress update interval is 15 seconds. And providers are not required to provide progress updates using the requested interval if it is seen as too demanding on the provider.

Security, Authorization, and Authentication

This specification does not address the question of how a provider should determine authorization for certain requests that are made using this protocol. It is up to the implementer of each harness to decide how and when to authorize various actions. Typically, XMPP itself provides the means to identify the requester (using its JID) and a provider can assume that the XMPP service has authenticated that entity.

Context

For normal harness requesters, there is no particular need to provide context information in their requests.

Harness providers, on the other hand, that use other harnesses as part of the processing of requests on their own harness are required to pass along context as described earlier in this document.

It may be tempting to include a lot of context information in the open-ended <details> portion of the context. But remember that this information will accumulate through a chain of such requests and, for this reason, implementers are discouraged from putting significant amounts of information in the details – except, perhaps, when they are placed in some kind of troubleshooting mode.

It is also strongly discouraged from having any implementation that depends upon information that may come into a provider within the context – as that will tend to create implementations that will only work in certain specific scenarios. Occasionally, there may be vendor-specific reasons why context information needs to flow opaquely through other foreign services using context, but generally this is discouraged.

11. XML Schemas

The following XML Schema Definition (XSD) describes all of the various elements that may be carried inside XMPP packets as part of this protocol.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:h="http://ntaforum.org/2011/harness"
           xmlns:xml="http://www.w3.org/XML/1998/namespace"
           targetNamespace="http://ntaforum.org/2011/harness"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">
    <xs:import namespace="http://www.w3.org/XML/1998/namespace" schemaLocation="xml-1998.xsd"/>
    <xs:element name="harness" type="h:harness"/>
    <xs:complexType name="harness">
        <xs:sequence>
            <xs:element name="supportedMode" type="h:sessionMode" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="name" type="xs:string" use="required"/>
    </xs:complexType>
    <xs:element name="query-harness">
        <xs:annotation>
            <xs:documentation>Information about a certain harness including descriptions of the
harness itself as well as the actions, events, and nested harnesses it
supports.</xs:documentation>
        </xs:annotation>
        <xs:complexType>
                <xs:complexContent>
```



```
                    <xs:extension base="h:harnessDecl">
                        <xs:sequence>
                            <xs:element name="subharness" minOccurs="0" maxOccurs="unbounded"
type="h:namespace"/>
                        </xs:sequence>
                        <xs:attribute name="harness" type="h:namespace" use="required"/>
                    </xs:extension>
                </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="open">
        <xs:annotation>
            <xs:documentation>A request to open a new session using a specified harness and a set
of parameters and other information consistent with the contract defined in query-
harness</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="h:requestType">
                    <xs:sequence>
                        <xs:element name="timestamp" type="xs:dateTime" minOccurs="0"/>
                        <xs:element name="activationRef" type="xs:string"/>
                    </xs:sequence>
                    <xs:attribute name="harness" type="h:namespace" use="required"/>
                    <xs:attribute name="mode" type="h:sessionMode" use="required"/>
                    <xs:attribute name="reportUserActivity" type="xs:boolean" use="optional"
default="true"/>
                    <xs:attribute ref="xml:lang" use="optional"/>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="request">
        <xs:annotation>
            <xs:documentation>A request to perform a certain action within the context of a
session previously established using open, along with parameters and other information affecting
the behavior of this request, consistent with the contract defined by the corresponding query-
harness.</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="h:requestType">
                    <xs:sequence>
                        <xs:element name="timestamp" type="xs:dateTime" minOccurs="0"/>
                        <xs:element name="action" type="h:action"/>
                    </xs:sequence>
                    <xs:attribute name="session" type="xs:Name" use="required"/>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="cancel">
        <xs:annotation>
            <xs:documentation>A request to cancel an action previously requested using a
request</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element name="timestamp" type="xs:dateTime" minOccurs="0"/>
            </xs:sequence>
            <xs:attribute name="session" type="xs:Name" use="required"/>
            <xs:attribute name="requestId" type="xs:string" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="progress">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="totalWork" type="xs:int"/>
                <xs:element name="remainingWork" type="xs:int"/>
                <xs:element name="status" type="xs:string" minOccurs="0"/>
                <xs:element name="timeRemaining" type="xs:duration" minOccurs="0"/>
```



```
                <xs:element name="timestamp" type="xs:dateTime" minOccurs="0"/>
            </xs:sequence>
            <xs:attribute name="session" type="xs:Name" use="required"/>
            <xs:attribute name="requestId" type="xs:string" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="response">
        <xs:annotation>
            <xs:documentation>A response returned corresponding to a request previously made.
The response, if populated, will conform to the details defined in the contract established via
query-harness for the action from the request.</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element name="result" type="h:result"/>
                <xs:element name="message" type="xs:string" minOccurs="0"/>
                <xs:element name="duration" type="xs:float" minOccurs="0"/>
                <xs:element name="item" type="h:responseItem" minOccurs="0"
maxOccurs="unbounded"/>
                <xs:element name="xmlItem" type="h:responseXmlItem" minOccurs="0"
maxOccurs="unbounded"/>
                <xs:element name="file" type="h:fileItem" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element name="group" type="h:responseGroup" minOccurs="0"
maxOccurs="unbounded"/>
                <xs:element name="timestamp" type="xs:dateTime" minOccurs="0"/>
</xs:sequence>
            <xs:attribute name="session" type="xs:Name" use="required"/>
            <xs:attribute name="requestId" type="xs:string" use="optional"/>
            <xs:attribute ref="xml:lang" use="optional"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="event">
        <xs:annotation>
            <xs:documentation>An autonomous message issued by the provider of a harness to the
originator of one of its sessions informing it about some event that has occurred.  It may also
contain additional information that must conform to the corresponding information in the query-
harness for the corresponding event.</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element name="timestamp" type="xs:dateTime"/>
                <xs:element name="item" type="h:responseItem" minOccurs="0"
maxOccurs="unbounded"/>
                <xs:element name="xmlItem" type="h:responseXmlItem" minOccurs="0"
maxOccurs="unbounded"/>
                <xs:element name="file" type="h:fileItem" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element name="group" type="h:responseGroup" minOccurs="0"
maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="session" type="xs:Name" use="required"/>
            <xs:attribute name="harness" type="h:namespace" use="required"/>
            <xs:attribute name="name" type="xs:Name" use="required"/>
            <xs:attribute ref="xml:lang" use="optional"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="close">
        <xs:annotation>
            <xs:documentation>A request to close a session previously opened using
open.</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element name="timestamp" type="xs:dateTime" minOccurs="0"/>
            </xs:sequence>
            <xs:attribute name="session" type="xs:Name" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="notify-close">
        <xs:annotation>
            <xs:documentation>A notification from the provider that the session has
closed</xs:documentation>
```



```
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element name="timestamp" type="xs:dateTime" minOccurs="0"/>
            </xs:sequence>
            <xs:attribute name="session" type="xs:Name" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="notify-action">
        <xs:annotation>
            <xs:documentation>Notification of information about an action that was performed and
the response it produced.</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element name="action" type="h:action"/>
                <xs:element name="started" type="xs:dateTime"/>
                <xs:element name="context" type="h:context" minOccurs="0"/>
                <xs:element name="requestParameter" type="h:requestParameter" minOccurs="0"
maxOccurs="unbounded"/>
                <xs:element name="requestXmlParameter" type="h:requestXmlParameter" minOccurs="0"
maxOccurs="unbounded"/>
                <xs:element name="requestFile" type="h:requestFile" minOccurs="0"
maxOccurs="unbounded"/>
                <xs:element name="requestGroup" type="h:requestGroup" minOccurs="0"
maxOccurs="unbounded"/>
                <xs:element name="result" type="h:result"/>
                <xs:element name="message" type="xs:string" minOccurs="0"/>
                <xs:element name="duration" type="xs:float" minOccurs="0"/>
                <xs:element name="responseItem" type="h:responseItem" minOccurs="0"
maxOccurs="unbounded"/>
                <xs:element name="responseXmlItem" type="h:responseXmlItem" minOccurs="0"
maxOccurs="unbounded"/>
                <xs:element name="responseFile" type="h:fileItem" minOccurs="0"
maxOccurs="unbounded"/>
                <xs:element name="responseGroup" type="h:responseGroup" minOccurs="0"
maxOccurs="unbounded"/>
                <xs:element name="timestamp" type="xs:dateTime" minOccurs="0"/>
</xs:sequence>
            <xs:attribute name="session" type="xs:Name" use="required"/>
            <xs:attribute ref="xml:lang" use="optional"/>
        </xs:complexType>
    </xs:element>
    <!-- The remainder of the file contains definitions for classes referenced above -->
        <xs:complexType name="harnessDecl">
        <xs:annotation>
            <xs:documentation>Information about a named set of actions and events that may
describe a harness</xs:documentation>
        </xs:annotation>
        <xs:complexContent>
            <xs:extension base="h:elementDecl">
                <xs:sequence>
                    <xs:element name="supercedes" type="h:namespace" minOccurs="0"/>
                    <xs:element name="author" type="xs:string"/>
                    <xs:element name="actionDecl" minOccurs="0" maxOccurs="unbounded">
                        <xs:complexType>
                            <xs:complexContent>
                                <xs:extension base="h:namedElementDecl">
                                    <xs:sequence>
                                        <xs:element name="parameter" type="h:parameterDecl"
minOccurs="0" maxOccurs="unbounded"/>
                                        <xs:element name="xmlParameter" type="h:xmlParameterDecl"
minOccurs="0" maxOccurs="unbounded"/>
                                        <xs:element name="file" type="h:fileDecl" minOccurs="0"
maxOccurs="unbounded"/>
                                        <xs:element name="group" type="h:requestGroupDecl"
minOccurs="0" maxOccurs="unbounded"/>
                                        <xs:element name="responseDecl" type="h:responseDecl"
minOccurs="0"/>
                                    </xs:sequence>
                                </xs:extension>
```



```
                        </xs:complexContent>
                    </xs:complexType>
                </xs:element>
                <xs:element name="eventDecl" minOccurs="0" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:complexContent>
                            <xs:extension base="h:responseDecl">
                                <xs:sequence>
                                    <xs:element name="description" type="xs:string"/>
                                </xs:sequence>
                                <xs:attribute name="name" type="xs:Name" use="required"/>
                            </xs:extension>
                        </xs:complexContent>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
            <xs:attribute ref="xml:lang" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<xs:complexType name="responseDecl">
    <xs:annotation>
        <xs:documentation>A declaration of the information that will be returned in response
to a request for a given action supported by the harness (or harness including support for the
given sub)</xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="item" type="h:responseItemDecl" minOccurs="0"
maxOccurs="unbounded"/>
        <xs:element name="xmlItem" type="h:responseXmlItemDecl" minOccurs="0"
maxOccurs="unbounded"/>
        <xs:element name="fileItem" type="h:fileItemDecl" minOccurs="0"
maxOccurs="unbounded"/>
        <xs:element name="group" type="h:responseGroupDecl" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="requestType">
    <xs:annotation>
        <xs:documentation>Information that is included along with a request that is made in
the context of a session using a given harness.  This information needs to conform to the
contract laid out in the query-harness for the corresponding action.</xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="context" type="h:context" minOccurs="0"/>
        <xs:element name="parameter" type="h:requestParameter" minOccurs="0"
maxOccurs="unbounded"/>
        <xs:element name="xmlParameter" type="h:requestXmlParameter" minOccurs="0"
maxOccurs="unbounded"/>
        <xs:element name="file" type="h:requestFile" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="group" type="h:requestGroup" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="parameterDecl">
    <xs:annotation>
        <xs:documentation>Declaration of the information that may be used to describe a
certain parameter associated with a certain action declaration on a certain harness
        </xs:documentation>
    </xs:annotation>
    <xs:complexContent>
        <xs:extension base="h:namedElementDeclWithMandatory">
            <xs:sequence>
                <xs:element name="datatype" type="h:dataType" default="string"
minOccurs="0"/>
                <xs:element name="units" type="xs:string" minOccurs="0"/>
                <xs:element name="default" type="xs:string" minOccurs="0"/>
                <xs:element name="masked" type="xs:boolean" default="false" minOccurs="0"/>
                <xs:element name="isMultiline" type="xs:boolean" default="false"
minOccurs="0"/>
                <xs:element name="allowedValue" type="h:allowedValueDecl" minOccurs="0"
maxOccurs="unbounded"/>
```



```
                    <xs:element name="allowedLength" type="h:allowedCountDecl" minOccurs="0"/>
                    <xs:element name="allowedCount" type="h:allowedCountDecl" minOccurs="0"/>
                    <xs:element name="allowedPattern" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
                    <xs:element name="allowedRange" type="h:allowedRangeDecl" minOccurs="0"
maxOccurs="unbounded"/>
                    <xs:element name="enablementValue" type="h:enablementValueDecl"
minOccurs="0"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="xmlParameterDecl">
        <xs:annotation>
            <xs:documentation>Declaration of the information that may be used to describe a
certain XML parameter -- i.e., a document fragment that may be included as part of a request to
perform a given action</xs:documentation>
        </xs:annotation>
        <xs:complexContent>
            <xs:extension base="h:namedElementDeclWithMandatory">
                <xs:sequence>
                    <xs:element name="element" type="xs:Name"/>
                    <xs:element name="xmlNamespace" type="xs:string"/>
                    <xs:element name="enablementValue" type="h:enablementValueDecl"
minOccurs="0"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="fileDecl">
        <xs:annotation>
            <xs:documentation>Declaration of the information that may be used to describe a file
that may be &amp;quot;attached&amp;quot; to the corresponding action request</xs:documentation>
        </xs:annotation>
        <xs:complexContent>
            <xs:extension base="h:namedElementDeclWithMandatory">
                <xs:sequence>
                    <xs:element name="allowedCount" type="h:allowedCountDecl" minOccurs="0"/>
                    <xs:element name="allowedFileExtension" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
                    <xs:element name="enablementValue" type="h:enablementValueDecl"
minOccurs="0"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="allowedValueDecl">
        <xs:simpleContent>
            <xs:extension base="xs:string">
                <xs:attribute name="label" type="xs:string"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
    <xs:complexType name="allowedRangeDecl">
        <xs:sequence>
            <xs:element name="min" type="xs:decimal" minOccurs="0"/>
            <xs:element name="max" type="xs:decimal" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="allowedCountDecl">
        <xs:sequence>
            <xs:element name="min" type="xs:int" minOccurs="0"/>
            <xs:element name="max" type="xs:int" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="enablementValueDecl">
        <xs:sequence>
            <xs:element name="parameter" type="xs:Name"/>
            <xs:element name="value" type="xs:string"/>
            <xs:element name="enableOn" type="h:enableOnType"/>
        </xs:sequence>
```



```
    </xs:complexType>
    <xs:complexType name="requestGroupDecl">
        <xs:annotation>
            <xs:documentation>Declaration of a named group of parameters that may be included in
a request for a certain action</xs:documentation>
        </xs:annotation>
        <xs:complexContent>
            <xs:extension base="h:namedElementDeclWithMandatory">
                <xs:sequence>
                    <xs:element name="allowedCount" type="h:allowedCountDecl" minOccurs="0"/>
                    <xs:element name="parameterKeyName" type="xs:Name" minOccurs="0"/>
                    <xs:element name="parameter" type="h:parameterDecl" minOccurs="0"
maxOccurs="unbounded"/>
                    <xs:element name="group" type="h:requestGroupDecl" minOccurs="0"
maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="responseItemDecl">
        <xs:annotation>
            <xs:documentation>Declaration of a certain specific item of information that may be
returned in the response to a certain action</xs:documentation>
        </xs:annotation>
        <xs:complexContent>
            <xs:extension base="h:namedElementDeclWithMandatory">
                <xs:sequence>
                    <xs:element name="default" type="xs:string" minOccurs="0"/>
                    <xs:element name="datatype" type="h:dataType" default="string"
minOccurs="0"/>
                    <xs:element name="units" type="xs:string" minOccurs="0"/>
                    <xs:element name="masked" type="xs:boolean" default="false" minOccurs="0"/>
                    <xs:element name="isMultiline" type="xs:boolean" default="false"
minOccurs="0"/>
                    <xs:element name="allowedValue" type="h:allowedValueDecl" minOccurs="0"
maxOccurs="unbounded"/>
                    <xs:element name="allowedCount" type="h:allowedCountDecl" minOccurs="0"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="responseXmlItemDecl">
        <xs:annotation>
            <xs:documentation>Declaration of a certain named XML document fragment that may be
returned in the response to a certain action</xs:documentation>
        </xs:annotation>
        <xs:complexContent>
            <xs:extension base="h:namedElementDeclWithMandatory">
                <xs:sequence>
                    <xs:element name="element" type="xs:Name"/>
                    <xs:element name="xmlNamespace" type="xs:string"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="fileItemDecl">
        <xs:annotation>
            <xs:documentation>Declaration of a certain named file that may be returned in the
response to a certain action</xs:documentation>
        </xs:annotation>
        <xs:complexContent>
            <xs:extension base="h:namedElementDeclWithMandatory">
                <xs:sequence>
                    <xs:element name="allowedCount" type="h:allowedCountDecl" minOccurs="0"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="responseGroupDecl">
        <xs:annotation>
```



```
            <xs:documentation>Declaration of a certain named group of items that may be returned
in the response to a certain action</xs:documentation>
        </xs:annotation>
        <xs:complexContent>
            <xs:extension base="h:namedElementDeclWithMandatory">
                <xs:sequence>
                    <xs:element name="allowedCount" type="h:allowedCountDecl" minOccurs="0"/>
                    <xs:element name="itemKeyName" type="xs:Name" minOccurs="0"/>
                    <xs:element name="item" type="h:responseItemDecl" minOccurs="0"
maxOccurs="unbounded"/>
                    <xs:element name="group" type="h:responseGroupDecl" minOccurs="0"
maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="responseItem">
        <xs:simpleContent>
            <xs:extension base="xs:string">
                <xs:attribute name="name" type="xs:Name" use="required"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
    <xs:complexType name="fileItem">
        <xs:sequence>
            <xs:element name="filename" type="xs:string"/>
        </xs:sequence>
        <xs:attribute name="name" type="xs:Name" use="required"/>
    </xs:complexType>
    <xs:complexType name="responseXmlItem">
        <xs:sequence>
            <xs:any processContents="skip"/>
        </xs:sequence>
        <xs:attribute name="name" type="xs:Name" use="required"/>
    </xs:complexType>
    <xs:complexType name="responseGroup">
        <xs:sequence>
            <xs:element name="item" type="h:responseItem" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="group" type="h:responseGroup" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="name" type="xs:Name" use="required"/>
    </xs:complexType>
    <xs:complexType name="action">
        <xs:simpleContent>
            <xs:extension base="h:actionName">
                <xs:attribute name="harness" type="h:namespace" use="required"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
    <xs:complexType name="context">
        <xs:sequence>
            <xs:element name="details" minOccurs="0">
                <xs:complexType>
                    <xs:sequence>
                        <xs:any processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="context" type="h:context" minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="entity" type="h:jid" use="optional"/>
        <xs:attribute name="harness" type="xs:string" use="optional"/>
        <xs:attribute name="session" type="xs:Name" use="optional"/>
        <xs:attribute name="requestId" type="xs:string" use="optional"/>
    </xs:complexType>
    <xs:complexType name="requestParameter">
        <xs:simpleContent>
            <xs:extension base="xs:string">
                <xs:attribute name="name" type="xs:Name" use="required"/>
            </xs:extension>
        </xs:simpleContent>
```



```
    </xs:complexType>
    <xs:complexType name="requestXmlParameter">
        <xs:sequence>
            <xs:any processContents="skip"/>
        </xs:sequence>
        <xs:attribute name="name" type="xs:Name" use="required"/>
    </xs:complexType>
    <xs:complexType name="requestFile">
        <xs:sequence>
            <xs:element name="filename" type="xs:string"/>
        </xs:sequence>
        <xs:attribute name="name" type="xs:Name" use="required"/>
    </xs:complexType>
    <xs:complexType name="requestGroup">
        <xs:sequence>
            <xs:element name="parameter" type="h:requestParameter" minOccurs="0"
maxOccurs="unbounded"/>
            <xs:element name="group" type="h:requestGroup" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="name" type="xs:Name" use="required"/>
    </xs:complexType>
    <xs:simpleType name="enableOnType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="equal"/>
            <xs:enumeration value="not_equal"/>
            <xs:enumeration value="pattern match"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="sessionMode">
        <xs:restriction base="xs:string">
            <xs:enumeration value="visible_and_interactive"/>
            <xs:enumeration value="visible and automated"/>
            <xs:enumeration value="invisible_and_automated"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="result">
        <xs:restriction base="xs:string">
            <xs:enumeration value="pass"/>
            <xs:enumeration value="fail"/>
            <xs:enumeration value="abort"/>
            <xs:enumeration value="pending"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="dataType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="string"/>
            <xs:enumeration value="integer"/>
            <xs:enumeration value="decimal"/>
            <xs:enumeration value="boolean"/>
            <xs:enumeration value="anyURI"/>
            <xs:enumeration value="dateTime"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="jid">
        <xs:restriction base="xs:string"/>
    </xs:simpleType>
    <xs:simpleType name="namespace">
        <xs:restriction base="xs:anyURI"/>
    </xs:simpleType>
    <xs:simpleType name="actionName">
        <xs:restriction base="xs:Name"/>
    </xs:simpleType>
    <xs:complexType name="namedElementDeclWithMandatory">
        <xs:complexContent>
            <xs:extension base="h:namedElementDecl">
                <xs:sequence>
                    <xs:element name="mandatory" type="xs:boolean" default="true" minOccurs="0"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
```



```
    <xs:complexType name="namedElementDecl">
        <xs:complexContent>
            <xs:extension base="h:elementDecl">
                <xs:attribute name="name" type="h:actionName" use="required"/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="elementDecl">
        <xs:sequence>
            <xs:element name="label" type="xs:string"/>
            <xs:element name="tooltip" type="xs:string" minOccurs="0"/>
            <xs:element name="description" type="xs:string" minOccurs="0"/>
            <xs:element name="helpURI" type="xs:anyURI" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

