



Web Services – Human Task (WS-HumanTask) Specification Version 1.1

Committee Draft 06 / Public Review Draft 01

04 November 2009

Specification URIs:

This Version:

<http://docs.oasis-open.org/bpel4people/ws-humantask-1.1-spec-cd-06.html>
<http://docs.oasis-open.org/bpel4people/ws-humantask-1.1-spec-cd-06.doc> (Authoritative format)
<http://docs.oasis-open.org/bpel4people/ws-humantask-1.1-spec-cd-06.pdf>

Previous Version:

N/A

Latest Version:

<http://docs.oasis-open.org/bpel4people/ws-humantask-1.1.html>
<http://docs.oasis-open.org/bpel4people/ws-humantask-1.1.doc>
<http://docs.oasis-open.org/bpel4people/ws-humantask-1.1.pdf>

Technical Committee:

OASIS BPEL4People TC

Chair:

Dave Ings, IBM

Editors:

Luc Clément, Active Endpoints, Inc.
Dieter König, IBM
Vinkesh Mehta, Deloitte Consulting LLP
Ralf Mueller, Oracle Corporation
Ravi Rangaswamy, Oracle Corporation
Michael Rowley, Active Endpoints, Inc.
Ivana Trickovic, SAP

Related work:

This specification is related to:

- WS-BPEL Extension for People (BPEL4People) Specification – Version 1.1 -
<http://docs.oasis-open.org/bpel4people/bpel4people-1.1.html>

Declared XML Namespaces:

htd – <http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803>
hta – <http://docs.oasis-open.org/ns/bpel4people/ws-humantask/api/200803>
htlt - <http://docs.oasis-open.org/ns/bpel4people/ws-humantask/leantask/api/200803>
htt – <http://docs.oasis-open.org/ns/bpel4people/ws-humantask/types/200803>

htc - <http://docs.oasis-open.org/ns/bpel4people/ws-humantask/context/200803>

htcp- <http://docs.oasis-open.org/ns/bpel4people/ws-humantask/protocol/200803>

htp - <http://docs.oasis-open.org/ns/bpel4people/ws-humantask/policy/200803>

Abstract:

The concept of human tasks is used to specify work which has to be accomplished by people. Typically, human tasks are considered to be part of business processes. However, they can also be used to design human interactions which are invoked as services, whether as part of a process or otherwise.

This specification introduces the definition of human tasks, including their properties, behavior and a set of operations used to manipulate human tasks. A coordination protocol is introduced in order to control autonomy and life cycle of service-enabled human tasks in an interoperable manner.

Status:

This document was last revised or approved by the OASIS WS-BPEL Extension for People Technical Committee on the above date. The level of approval is also listed above. Check the "Latest Version" or "Latest Approved Version" location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at <http://www.oasis-open.org/committees/bpel4people/>.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (<http://www.oasis-open.org/committees/bpel4people/ipr.php>).

The non-normative errata page for this specification is located at <http://www.oasis-open.org/committees/bpel4people/>.

Notices

Copyright © OASIS® 2009. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The names "OASIS", are trademarks of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see <http://www.oasis-open.org/who/trademark.php> for above guidance.

Table of Contents

1	Introduction	7
1.1	Terminology	7
1.2	Normative References	8
1.3	Non-Normative References	9
1.4	Conformance Targets	9
1.5	Overall Architecture	9
2	Language Design	16
2.1	Dependencies on Other Specifications	16
2.1.1	Namespaces Referenced	16
2.2	Language Extensibility	16
2.3	Overall Language Structure	17
2.3.1	Syntax	17
2.3.2	Properties	17
2.4	Default use of XPath 1.0 as an Expression Language	19
3	Concepts	20
3.1	Generic Human Roles	20
3.2	Composite Tasks and Sub Tasks	21
3.2.1	Composite Tasks by Definition	21
3.2.2	Composite Tasks Created Adhoc at Runtime	21
3.3	Routing Patterns	21
3.4	Relationship of Composite Tasks and Routing Patterns	22
3.5	Assigning People	22
3.5.1	Using Logical People Groups	23
3.5.2	Using Literals	24
3.5.3	Using Expressions	25
3.5.4	Data Type for Organizational Entities	26
3.5.5	Subtasks	26
3.6	Task Rendering	27
3.7	Lean Tasks	27
3.8	Task Instance Data	28
3.8.1	Presentation Data	28
3.8.2	Context Data	28
3.8.3	Operational Data	28
3.8.4	Data Types for Task Instance Data	30
3.8.5	Sub Tasks	33
4	Human Tasks	35
4.1	Overall Syntax	35
4.2	Properties	36
4.3	Presentation Elements	37
4.4	Task Possible Outcomes	40

4.5	Elements for Rendering Tasks	40
4.6	Elements for Composite Tasks	41
4.7	Elements for People Assignment	42
4.7.1	Routing Patterns	43
4.8	Completion Behavior	45
4.8.1	Completion Conditions.....	46
4.8.2	Result Construction from Parallel Subtasks	47
4.9	Elements for Handling Timeouts and Escalations.....	51
4.10	Human Task Behavior and State Transitions.....	58
4.10.1	Normal processing of a Human Task	58
4.10.2	Releasing a Human Task	59
4.10.3	Delegating or Forwarding a Human Task.....	59
4.10.4	Sub Task Event Propagation	59
4.11	History of a Human Task.....	60
4.11.1	Task Event Types and Data	61
4.11.2	Retrieving the History	63
5	Lean Tasks	66
5.1	Overall Syntax	66
5.2	Properties	66
5.3	Message Schema.....	66
5.4	Example: ToDoTask.....	68
6	Notifications	69
6.1	Overall Syntax	69
6.2	Properties	70
6.3	Notification Behavior and State Transitions	70
7	Programming Interfaces.....	71
7.1	Operations for Client Applications	71
7.1.1	Participant Operations	71
7.1.2	Simple Query Operations	83
7.1.3	Advanced Query Operation	87
7.1.4	Administrative Operations.....	90
7.1.5	Operation Authorizations	91
7.2	XPath Extension Functions	93
8	Interoperable Protocol for Advanced Interaction with Human Tasks	100
8.1	Human Task Coordination Protocol Messages.....	102
8.2	Protocol Messages	103
8.2.1	Protocol Messages Received by a Task Parent.....	103
8.2.2	Protocol Messages Received by a Task	103
8.3	WSDL of the Protocol Endpoints.....	103
8.3.1	Protocol Endpoint of the Task Parent.....	103
8.3.2	Protocol Endpoint of the Task.....	104
8.4	Providing Human Task Context.....	104
8.4.1	SOAP Binding of Human Task Context.....	104

8.4.2	Overriding Task Definition People Assignments	105
8.5	Human Task Policy Assertion	106
9	Task Parent Interactions with Lean Tasks.....	107
9.1	Operations for Task Parent Applications.....	107
9.2	Lean Task Interactions	107
9.2.1	Register a Lean Task Definition.....	107
9.2.2	Unregister a Lean Task Definition	108
9.2.3	List Lean Task Definitions.....	108
9.2.4	Create a Lean Task	109
9.2.5	Endpoints for Lean Task Operations	110
10	Providing Callback Information for Human Tasks.....	112
10.1	EPR Information Model Extension	112
10.2	XML Infoset Representation.....	112
10.3	Message Addressing Properties	114
10.4	SOAP Binding.....	115
11	Security Considerations	118
12	Conformance.....	119
A.	Portability and Interoperability Considerations	120
B.	WS-HumanTask Language Schema	121
C.	WS-HumanTask Data Types Schema.....	136
D.	WS-HumanTask Client API Port Type	145
E.	WS-HumanTask Parent API Port Type.....	184
F.	WS-HumanTask Protocol Handler Port Types	190
G.	WS-HumanTask Context Schema	192
H.	WS-HumanTask Policy Assertion Schema.....	195
I.	Sample.....	196
J.	Acknowledgements.....	206
K.	Non-Normative Text.....	208
L.	Revision History	209

1 Introduction

Human tasks, or briefly *tasks* enable the integration of human beings in service-oriented applications. This document provides a notation, state diagram and API for human tasks, as well as a coordination protocol that allows interaction with human tasks in a more service-oriented fashion and at the same time controls tasks' autonomy. The document is called Web Services Human Task (abbreviated to WS-HumanTask for the rest of this document).

Human tasks are services "implemented" by people. They allow the integration of humans in service-oriented applications. A human task has two interfaces. One interface exposes the service offered by the task, like a translation service or an approval service. The second interface allows people to deal with tasks, for example to query for human tasks waiting for them, and to work on these tasks.

A human task has people assigned to it. These assignments define who should be allowed to play a certain role on that task. Human tasks might be assigned to people in a well-defined order. This includes assignments in a specific sequence and or parallel assignment to a set of people or any combination of both. Human tasks may also specify how task metadata should be rendered on different devices or applications making them portable and interoperable with different types of software. Human tasks can be defined to react to timeouts, triggering an appropriate escalation action.

This also holds true for *notifications*. A notification is a special type of human task that allows the sending of information about noteworthy business events to people. Notifications are always one-way, i.e., they are delivered in a fire-and-forget manner: The sender pushes out notifications to people without waiting for these people to acknowledge their receipt.

Let us take a look at an example, an approval task. Such a human task could be involved in a mortgage business process. After the data of the mortgage has been collected, and, if the value exceeds some amount, a manual approval step is required. This can be implemented by invoking an approval service implemented by the approval task. The invocation of the service by the business process creates an instance of the approval task. As a consequence this task pops up on the task list of the approvers. One of the approvers will claim the task, evaluate the mortgage data, and eventually complete the task by either approving or rejecting it. The output message of the task indicates whether the mortgage has been approved or not. All of the above is transparent to the caller of the task (a business process in this example).

The goal of this specification is to enable portability and interoperability:

- Portability - The ability to take human tasks and notifications created in one vendor's environment and use them in another vendor's environment.
- Interoperability - The capability for multiple components (task infrastructure, task list clients and applications or processes with human interactions) to interact using well-defined messages and protocols. This enables combining components from different vendors allowing seamless execution.

Out of scope of this specification is how human tasks and notifications are deployed or monitored. Usually people assignment is accomplished by performing queries on a people directory which has a certain organizational model. The mechanism determining how an implementation evaluates people assignments, as well as the structure of the data in the people directory is out of scope.

1.1 Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC 2119].

46 1.2 Normative References

47 [RFC 1766]

48 Tags for the Identification of Languages, RFC 1766, available via
49 <http://www.ietf.org/rfc/rfc1766.txt>

50 [RFC 2046]

51 Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types, RFC 2046, available via
52 <http://www.isi.edu/in-notes/rfc2046.txt> (or <http://www.iana.org/assignments/media-types/>)

53 [RFC 2119]

54 Key words for use in RFCs to Indicate Requirement Levels, RFC 2119, available via
55 <http://www.ietf.org/rfc/rfc2119.txt>

56 [RFC 2396]

57 Uniform Resource Identifiers (URI): Generic Syntax, RFC 2396, available via
58 <http://www.faqs.org/rfcs/rfc2396.html>

59 [RFC 3066]

60 Tags for the Identification of Languages, H. Alvestrand, IETF, January 2001, available via
61 <http://www.isi.edu/in-notes/rfc3066.txt>

62 [WSDL 1.1]

63 Web Services Description Language (WSDL) Version 1.1, W3C Note, available via
64 <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>

65 [WS-Addr-Core]

66 Web Services Addressing 1.0 - Core, W3C Recommendation, May 2006, available via
67 <http://www.w3.org/TR/ws-addr-core>

68 [WS-Addr-SOAP]

69 Web Services Addressing 1.0 – SOAP Binding, W3C Recommendation, May 2006, available via
70 <http://www.w3.org/TR/ws-addr-soap>

71 [WS-Addr-WSDL]

72 Web Services Addressing 1.0 – WSDL Binding, W3C Working Draft, February 2006, available via
73 <http://www.w3.org/TR/ws-addr-wsdl>

74 [WS-C]

75 OASIS Standard, “Web Services Coordination (WS-Coordination) Version 1.1”, 16 April 2007,
76 <http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.1-spec/wstx-wscoor-1.1-spec.html>

77 [WS-Policy]

78 Web Services Policy 1.5 - Framework, W3C Candidate Recommendation 30 March 2007,
79 available via <http://www.w3.org/TR/ws-policy/>

80 [WS-PolAtt]

81 Web Services Policy 1.5 - Attachment, W3C Candidate Recommendation 30 March 2007,
82 available via <http://www.w3.org/TR/2007/CR-ws-policy-attach-20070330/>

83 [XML Infoset]

84 XML Information Set, W3C Recommendation, available via [http://www.w3.org/TR/2001/REC-xml-](http://www.w3.org/TR/2001/REC-xml-infoset-20011024/)
85 [infoset-20011024/](http://www.w3.org/TR/2001/REC-xml-infoset-20011024/)

86 [XML Namespaces]

87 Namespaces in XML 1.0 (Second Edition), W3C Recommendation, available via
88 <http://www.w3.org/TR/REC-xml-names/>

89 [XML Schema Part 1]

90 XML Schema Part 1: Structures, W3C Recommendation, October 2004, available via
91 <http://www.w3.org/TR/xmlschema-1/>

92 **[XML Schema Part 2]**

93 XML Schema Part 2: Datatypes, W3C Recommendation, October 2004, available via
94 <http://www.w3.org/TR/xmlschema-2/>

95 **[XMLSpec]**

96 XML Specification, W3C Recommendation, February 1998, available via
97 <http://www.w3.org/TR/1998/REC-xml-19980210>

98 **[XPath 1.0]**

99 XML Path Language (XPath) Version 1.0, W3C Recommendation, November 1999, available via
100 <http://www.w3.org/TR/1999/REC-xpath-19991116>

101 **1.3 Non-Normative References**

102 There are no non-normative references made by this specification.

103 **1.4 Conformance Targets**

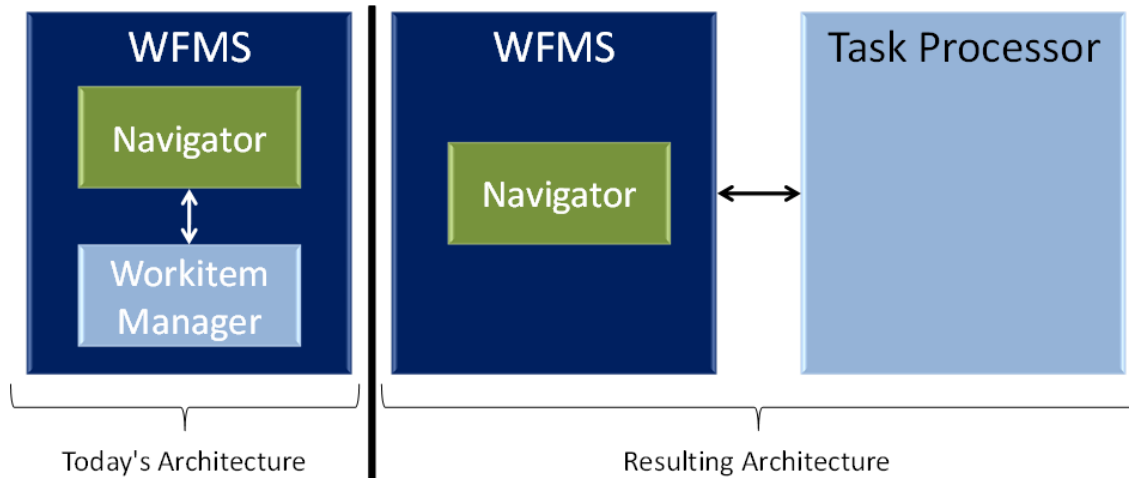
104 The following conformance targets are defined as part of this specification

- 105 • WS-HumanTask Definition
106 A WS-HumanTask Definition is any artifact that complies with the human interaction schema and
107 additional constraints defined in this document.
- 108 • WS-HumanTask Processor
109 A WS-HumanTask Processor is any implementation that accepts a WS-HumanTask definition
110 and executes the semantics as defined in this document.
- 111 • WS-HumanTask Parent
112 A WS-HumanTask Parent is any implementation that supports the Interoperable Protocol for
113 Advanced Interactions with Human Tasks as defined in this document.
- 114 • WS-HumanTask Client
115 A WS-HumanTask Client is any implementation that uses the Programming Interfaces of the
116 WS-HumanTask Processor.

117 **1.5 Overall Architecture**

118 One of the motivations of WS-HumanTask was an increasingly important need to support the ability to
119 allow any application to create human tasks in a service-oriented manner. Human tasks had traditionally
120 been created by tightly-coupled workflow management systems (WFMS). In such environments the
121 workflow management system managed the entirety of a task's lifecycle, an approach that did not allow
122 the means to directly affect a task's lifecycle outside of the workflow management environment (other
123 than for a human to actually carry out the task). Particularly significant was an inability to allow
124 applications to create a human task in such tightly coupled environments.

125



126
127

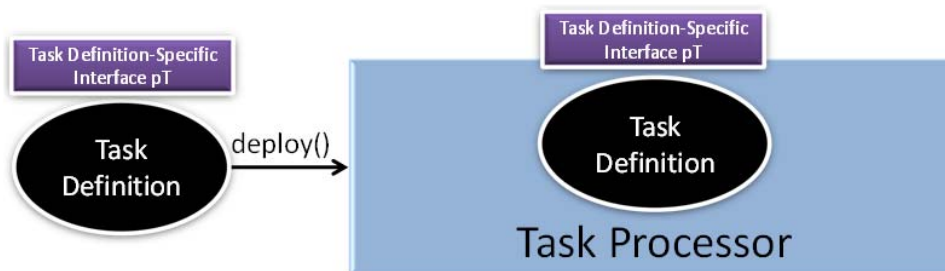
Figure 1- Architectural Impact of WS-HumanTask on Workflow Management Systems

128 The component within a WFMS typically responsible for managing a task's lifecycle (aka workitem) is
129 called a *Workitem Manager*. An example of such an environment is depicted on the left portion of Figure
130 1. The right portion of the figure depicts how significant a change of architecture WS-HumanTask
131 represents. Using this approach, the WFMS no longer incorporates a workitem manager but rather
132 interacts with a *Task Processor*. In this architecture the Task Processor is a separate, standalone
133 component exposed as a service, allowing any requestor to create tasks and interact with tasks. It is the
134 Task Processor's role to manage its tasks' lifecycle and to provide the means to "work" on tasks.

135 Conversely, by separating the Task Processor from the WFMS tasks can be used in the context of a
136 WFMS or any other WS-HumanTask application (also referred to as the *Task Parent*). A (special) case of
137 a business process acting as a Task Parent of a human task is described by the BPEL4People
138 specification.

139 WS-HumanTask tasks are assumed to have an interface. The interface of a task is represented as an
140 application-dependent port type referred to as its *Task Definition specific interface* (or *interface* for short –
141 see section 4.2). In order to create task instances (or *tasks* for short) managed by a particular Task
142 Processor, a port implementing the port type corresponding to a task needs to be deployed into the Task
143 Processor before it can be invoked. See Figure 2 depicting a Task Definition associated with a port type
144 (pT).

145



146
147

Figure 2 - Task Definitions Deployed in Task Processor

148 Once a task is available on the task processor any requestor can create task instances and interact with
149 them. The requestor that creates a task is referred to as the *Task Parent*. A task instance is created by
150 invoking an operation of the port type representing the interface of the task to be created. Typically port
151 types expose a single operation. Where more than one operation is defined, which operation of the port
152 type to be used to create a task is outside the scope of WS-HumanTask.

153

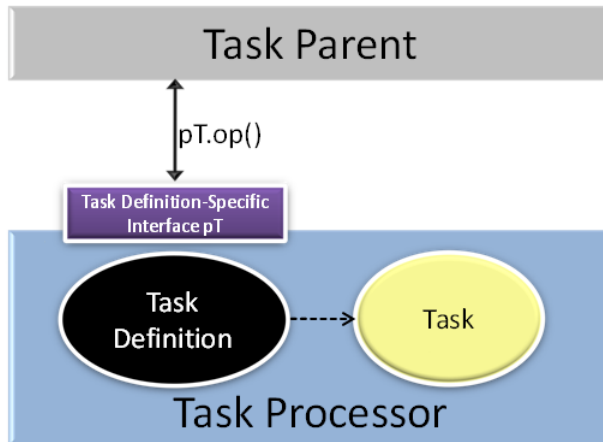


Figure 3 - Instantiating Tasks

154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175

In workflow environments the lifecycle of a task is typically dependent on the workflow system - i.e. tasks have to give up some of their autonomy. For example when a workflow is terminated prematurely, task initiated by that workflow should not be allowed to continue - the corresponding efforts to continue the work of the task would otherwise be wasted. To automate the corresponding behavior ensuring that the lifecycle of a Task Parent and the lifecycles of its initiated tasks are tightly coupled, WS-HumanTask uses the WS-Coordination specification as its coordination framework. This requires the definition of a coordination protocol following a particular behavior (see section 8). This is depicted by Figure 4.

When the Task Parent creates a task using the specific operation $op()$ of a port of port type pT , coordination context information is passed by the Task Parent to the environment hosting that port. Like any other WS-Coordination compliant coordination context, it contains the endpoint reference of (i.e. a "pointer" to) the coordinator to be used by the recipient of the context to register the corresponding coordination type. Note that for simplicity we assume in Figure 4 that the Task Processor itself is this recipient of the context information. Upon reception of the coordination context the Task Processor will register with the coordinator, implying that it passes the endpoint reference of its protocol handler to the coordinator (see section 8). In turn it will receive the endpoint reference of the protocol handler of the Task Parent. Similarly, for simplicity we assume in Figure 4 that the task parent provides its protocol handler. From that point on a coordination channel is established between the Task Parent and the Task Processor to exchange protocol messages allowing the coupling of the lifecycles of a task with its Task Parent. Section 4.10 describes the lifecycle of a task in more detail.

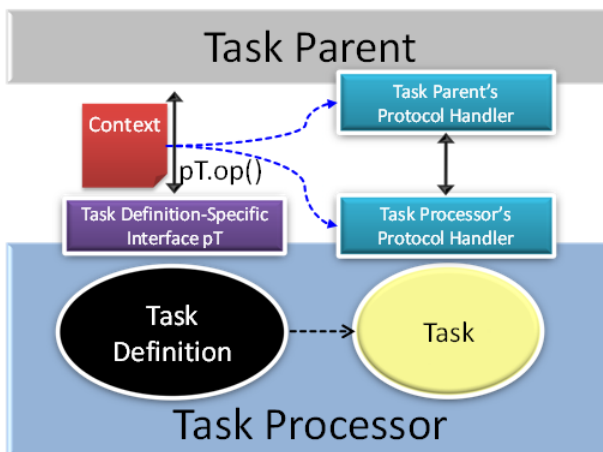
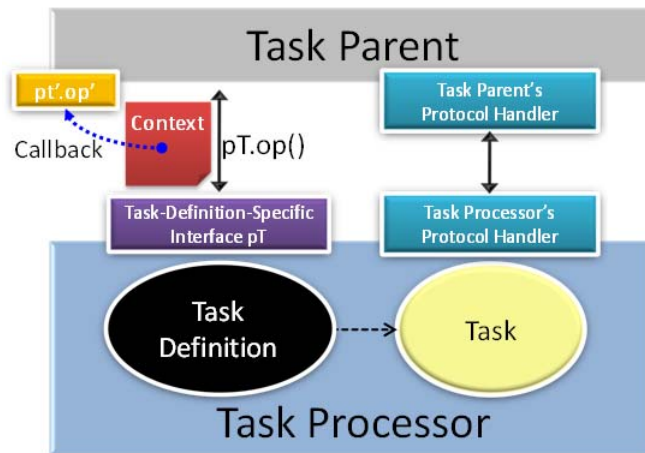


Figure 4 - Establishing a Protocol Channel

176
177
178

179 Most often tasks are long running in nature and will be invoked in an asynchronous manner. Thus, the
 180 Task Parent will kick-off the task and expects the result of the task to be returned at a later point in time.
 181 In order to allow the ability to pass the results back, the Task Processor needs to know where to send
 182 these results. For this purpose the context is extended with additional metadata that specifies the
 183 endpoint reference to be used to pass the result to, as well as the operation of the endpoint to be used by
 184 the Task Processor. Figure 5 depicts this by showing that the context contains information pointing to a
 185 port of port type *pt'* and specifying the name of the operation *op'* to be used on that port for returning
 186 results. Note that this behavior is compliant to WS-Addressing.
 187



188
 189 **Figure 5 - Passing Callback Information for Long Running Tasks**
 190

191 Finally, a Task Parent application invoking an operation implemented by a task is allowed to pass
 192 additional data along with the request message. This data is called the *human task context* and allows the
 193 ability to override some of the *Task Definition's* elements. Conversely, a human task context is also
 194 passed back with the response message, propagating information from the completed task to the Task
 195 Parent application, such as the task outcome or the task's actual people assignments.

196 Once a task is created it can be presented to its (potential) owners to be claimed and worked on. For that
 197 purpose another type of application called a *Task Client* is typically used. A Task Client presents to each
 198 of its users the tasks available to them. Users can then decide to claim the task to carry out the work
 199 associated with it. Other functions typically offered by a Task Client include the ability to skip a task, to
 200 add comments or attachments to a task, to nominate other users to perform the task and that like. In
 201 order to enable a Task Client to perform such functions on tasks, WS-HumanTask specifies the *task client*
 202 *interface* required to be implemented by Task Processor to support Task Clients (see section 7.1). Figure
 203 6 depicts the resultant architecture stemming from the introduction of Task Clients.

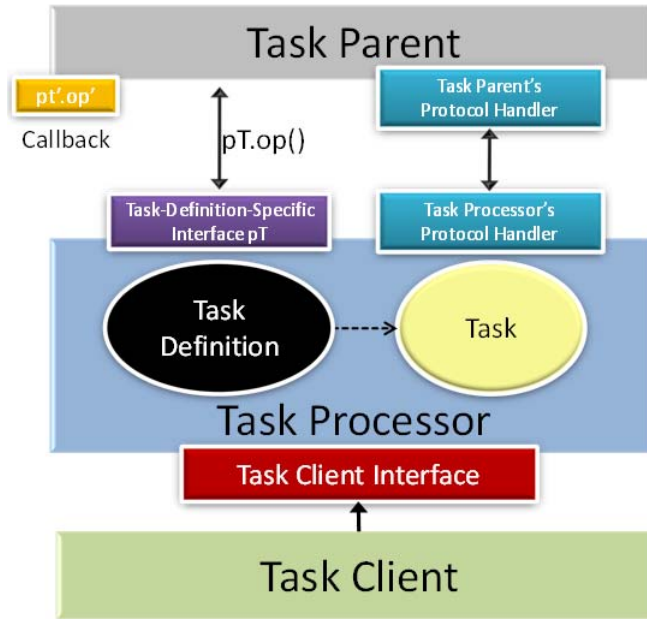
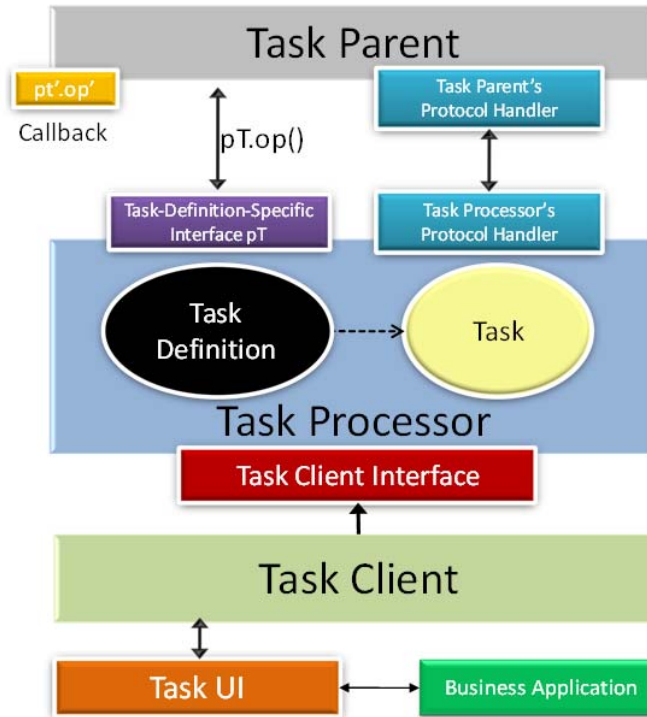


Figure 6 - Task List Client and Corresponding Interface

204
205
206

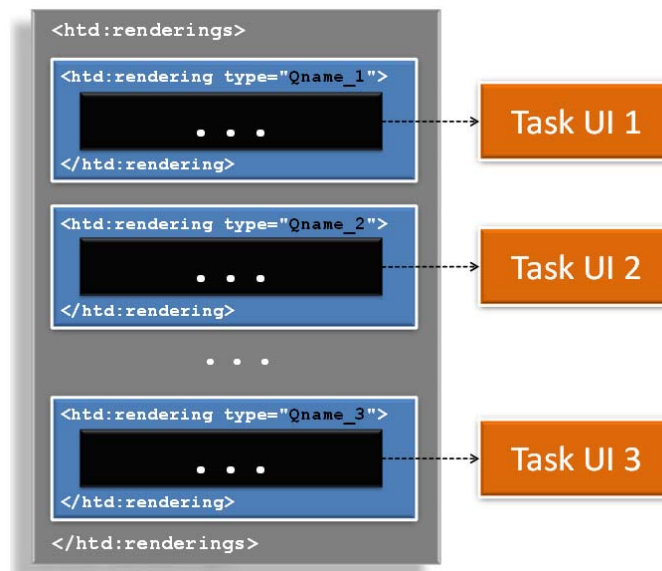
207 Once a user selects a task using his or her Task Client the user interface associated with the task is
 208 rendered allowing the user to view application-specific information pertaining to the task. WS-HumanTask
 209 does not specify such rendering but provides the means using a *container* to provide rendering hints to
 210 Task Clients. A Task Client in turn uses this information to construct or initiate the construction of the user
 211 interface of the task - the details how this is achieved are out of scope of WS-HumanTask. In the case of
 212 Lean Tasks, that rendering may be generated by the Task Processor. From the perspective of the Task
 213 Client, the fact the task is a Lean Task need not be apparent. Furthermore, the task may require the use
 214 of business applications to complete the task. Again the use of such business applications is out of scope
 215 of WS-HumanTask but such applications and their use are nonetheless important to the overall
 216 architecture depicted in Figure 7.



217
218

Figure 7 - Overall Architecture of a Human Task Infrastructure

219 The container referred to above for rendering a task's information is a task's `<rendering>` element (see
220 section 4.4). A rendering element specifies its type, expressed as a QName that denotes the kind of
221 rendering mechanism to use to generate the user interface for the task. All information actually needed to
222 create the user interface of the task is provided by the elements nested within the task's rendering
223 element (see Figure 8). The nested elements may also provide information about a business application
224 required to complete the task and other corresponding parameters.

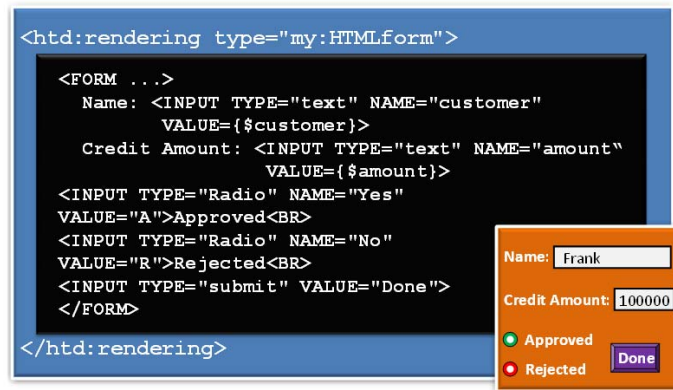


225
226

Figure 8 - Potential Renderings of a Task

227 For example Figure 9 depicts a rendering of type `my:HTMLform`. Its QName denotes that HTML forms
228 processing capabilities is needed to render the corresponding user interface of the task enclosing this
229 rendering. The nested element of the `my:HTMLform` rendering contains the actual HTML form to be

230 rendered. The example further assumes that the forms processor understands the {\$...} notation (see
231 section 4.3) to provide values from the task input as data presented in the form.



232
233

Figure 9 - Sample Rendering of a Task

234 A task may have different renderings associated with it. This allows the ability for a task to be rendered by
235 different access mechanisms or adapt to user preferences for example. How information is rendered is
236 out of scope of the WS-HumanTask specification.

237 2 Language Design

238 The language introduces a grammar for describing human tasks and notifications. Both design time
239 aspects, such as task properties and notification properties, and runtime aspects, such as task states and
240 events triggering transitions between states are covered by the language. Finally, it introduces a
241 programming interface which can be used by applications involved in the life cycle of a task to query task
242 properties, execute the task, or complete the task. This interface helps to achieve interoperability between
243 these applications and the task infrastructure when they come from different vendors.

244 The language provides an extension mechanism that can be used to extend the definitions with additional
245 vendor-specific or domain-specific information.

246 Throughout this specification, WSDL and schema elements may be used for illustrative or convenience
247 purposes. However, in a situation where those elements or other text within this document contradict the
248 separate WS-HumanTask, WSDL or schema files, it is those files that have precedence and not this
249 document.

250 2.1 Dependencies on Other Specifications

251 WS-HumanTask utilizes the following specifications:

- 252 • WSDL 1.1
- 253 • XML Schema 1.0
- 254 • XPath 1.0
- 255 • WS-Addressing 1.0
- 256 • WS-Coordination 1.1
- 257 • WS-Policy 1.5

258 2.1.1 Namespaces Referenced

259 WS-HumanTask references these namespaces:

- 260 • **wsa** – <http://www.w3.org/2005/08/addressing>
- 261 • **wSDL** – <http://schemas.xmlsoap.org/wSDL/>
- 262 • **wsp** – <http://www.w3.org/ns/ws-policy>
- 263 • **xsd** – <http://www.w3.org/2001/XMLSchema>

264 2.2 Language Extensibility

265 The WS-HumanTask extensibility mechanism allows:

- 266 • Attributes from other namespaces to appear on any WS-HumanTask element
- 267 • Elements from other namespaces to appear within WS-HumanTask elements

268 Extension attributes and extension elements **MUST NOT** contradict the semantics of any attribute or
269 element from the WS-HumanTask namespace. For example, an extension element could be used to
270 introduce a new task type.

271 The specification differentiates between mandatory and optional extensions (the section below explains
272 the syntax used to declare extensions). If a mandatory extension is used, a compliant implementation has
273 to understand the extension. If an optional extension is used, a compliant implementation can ignore the
274 extension.

275 2.3 Overall Language Structure

276 *Human interactions* subsume both human tasks and notifications. While human tasks and notifications
277 are described in subsequent sections, this section explains the overall structure of human interactions
278 definition.

279 2.3.1 Syntax

```
280 <htd:humanInteractions  
281   xmlns:htd="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803"  
282   xmlns:xsd="http://www.w3.org/2001/XMLSchema"  
283   xmlns:tns="anyURI"  
284   targetNamespace="anyURI"  
285   expressionLanguage="anyURI"?  
286   queryLanguage="anyURI"?>  
287  
288   <htd:extensions?>  
289     <htd:extension namespace="anyURI" mustUnderstand="yes|no"/>+  
290   </htd:extensions>  
291  
292   <htd:import namespace="anyURI"?  
293     location="anyURI"?  
294     importType="anyURI" />+*  
295  
296   <htd:logicalPeopleGroups?>  
297     <htd:logicalPeopleGroup name="NCName" reference="QName"?>+  
298       <htd:parameter name="NCName" type="QName" />+*  
299     </htd:logicalPeopleGroup>  
300   </htd:logicalPeopleGroups>  
301  
302   <htd:tasks?>  
303     <htd:task name="NCName">+  
304       ...  
305     </htd:task>  
306   </htd:tasks>  
307  
308   <htd:notifications?>  
309     <htd:notification name="NCName">+  
310       ...  
311     </htd:notification>  
312   </htd:notifications>  
313 </htd:humanInteractions>
```

314 2.3.2 Properties

315 The <humanInteractions> element has the following properties:

- 316 • `expressionLanguage`: This attribute specifies the expression language used in the enclosing
317 elements. The default value for this attribute is `urn:ws-ht:sublang:xpath1.0` which
318 represents the usage of XPath 1.0 within human interactions definition. A WS-HumanTask
319 Definition that uses expressions MAY override the default expression language for individual
320 expressions. A WS-HumanTask Processor MUST support the use of XPath 1.0 as the expression
321 language.
- 322 • `queryLanguage`: This attribute specifies the query language used in the enclosing elements.
323 The default value for this attribute is `urn:ws-ht:sublang:xpath1.0` which represents the
324 usage of XPath 1.0 within human interactions definition. A WS-HumanTask Definition that use

325 query expressions MAY override the default query language for individual query expressions. A
326 WS-HumanTask Processor MUST support the use of XPath 1.0 as the query language.

327 • **extensions**: This element is used to specify namespaces of WS-HumanTask extension
328 attributes and extension elements. The element is optional. If present, it MUST include at least
329 one extension element. The `<extension>` element is used to specify a namespace of WS-
330 HumanTask extension attributes and extension elements, and indicate whether they are
331 mandatory or optional. Attribute `mustUnderstand` is used to specify whether the extension must
332 be understood by a compliant implementation. If the attribute has value "yes" the extension is
333 mandatory. Otherwise, the extension is optional. If a WS-HumanTask Processor does not support
334 one or more of the extensions with `mustUnderstand="yes"`, then the human interactions definition
335 MUST be rejected. A WS-HumanTask Processor MAY ignore optional extensions. A WS-
336 HumanTask Definition MAY declare optional extensions. The same extension URI MAY be
337 declared multiple times in the `<extensions>` element. If an extension URI is identified as
338 mandatory in one `<extension>` element and optional in another, then the mandatory semantics
339 have precedence and MUST be enforced by a WS-HumanTask Processor. The extension
340 declarations in an `<extensions>` element MUST be treated as an unordered set.

341 • **import**: This element is used to declare a dependency on external WS-HumanTask and WSDL
342 definitions. Zero or more `<import>` elements MAY appear as children of the
343 `<humanInteractions>` element.

344 The `namespace` attribute specifies an absolute URI that identifies the imported definitions. This
345 attribute is optional. An `<import>` element without a `namespace` attribute indicates that external
346 definitions are in use which are not namespace-qualified. If a `namespace` is specified then the
347 imported definitions MUST be in that namespace. If no `namespace` is specified then the imported
348 definitions MUST NOT contain a `targetNamespace` specification. The `namespace`
349 `http://www.w3.org/2001/XMLSchema` is imported implicitly. Note, however, that there is no
350 implicit XML Namespace prefix defined for `http://www.w3.org/2001/XMLSchema`.

351 The `location` attribute contains a URI indicating the location of a document that contains
352 relevant definitions. The `location` URI MAY be a relative URI, following the usual rules for
353 resolution of the URI base [XML Base, RFC 2396]. The `location` attribute is optional. An
354 `<import>` element without a `location` attribute indicates that external definitions are used by
355 the human interactions definition but makes no statement about where those definitions can be
356 found. The `location` attribute is a hint and a WS-HumanTask Processor is not required to
357 retrieve the document being imported from the specified location.

358 The mandatory `importType` attribute identifies the type of document being imported by
359 providing an absolute URI that identifies the encoding language used in the document. The value
360 of the `importType` attribute MUST be set to `http://docs.oasis-`
361 `open.org/ns/bpel4people/ws-humantask/200803` when importing human interactions
362 definitions, to `http://schemas.xmlsoap.org/wsdl/` when importing WSDL 1.1 documents
363 or to `http://www.w3.org/2001/XMLSchema` when importing XML Schema documents.

364 According to these rules, it is permissible to have an `<import>` element without `namespace` and
365 `location` attributes, and only containing an `importType` attribute. Such an `<import>` element
366 indicates that external definitions of the indicated type are in use that are not namespace-
367 qualified, and makes no statement about where those definitions can be found.

368 A WS-HumanTask Definition MUST import all other WS-HumanTask definitions, WSDL
369 definitions, and XML Schema definitions it uses. In order to support the use of definitions from
370 namespaces spanning multiple documents, a WS-HumanTask Definition MAY include more than
371 one import declaration for the same `namespace` and `importType`, provided that those
372 declarations include different location values. `<import>` elements are conceptually unordered. A
373 WS-HumanTask Processor MUST reject the imported documents if they contain conflicting
374 definitions of a component used by the imported WS-HumanTask Definition.

375 Documents (or namespaces) imported by an imported document (or namespace) MUST NOT be
376 transitively imported by a WS-HumanTask Processor. In particular, this means that if an external
377 item is used by a task enclosed in the WS-HumanTask Definition, then a document (or
378 namespace) that defines that item MUST be directly imported by the WS-HumanTask Definition.
379 This requirement does not limit the ability of the imported document itself to import other
380 documents or namespaces.

- 381 • *logicalPeopleGroups*: This element specifies a set of logical people groups. The element is
382 optional. If present, it MUST include at least one *logicalPeopleGroup* element. The set of logical
383 people groups MUST contain only those logical people groups that are used in the
384 *humanInteractions* element, and enclosed human tasks and notifications. The
385 *logicalPeopleGroup* element has the following attributes. The *name* attribute specifies the name
386 of the logical people group. The name MUST be unique among the names of all
387 *logicalPeopleGroups* defined within the *humanInteractions* element. The *reference* attribute is
388 optional. In case a logical people group used in the *humanInteractions* element is defined in an
389 imported WS-HumanTask definition, the reference attribute MUST be used to specify the logical
390 people group. The *parameter* element is used to pass data needed for people query evaluation.
- 391 • *tasks*: This element specifies a set of human tasks. The element is optional. If present, it MUST
392 include at least one *<task>* element. The syntax and semantics of the *<task>* element are
393 introduced in section 4 “Human Tasks”.
- 394 • *notifications*: This element specifies a set of notifications. The element is optional. If
395 present, it MUST include at least one *<notification>* element. The syntax and semantics of the
396 *<notification>* element are introduced in section 6 “Notifications”.
- 397 • Element *humanInteractions* MUST NOT be empty, that is it MUST include at least one element.

398 All elements in WS-HumanTask Definition MAY use the element *<documentation>* to provide annotation
399 for users. The content could be a plain text, HTML, and so on. The *<documentation>* element is optional
400 and has the following syntax:

```
401 <htd:documentation xml:lang="xsd:language">  
402   ...  
403 </htd:documentation>
```

404 2.4 Default use of XPath 1.0 as an Expression Language

405 The XPath 1.0 specification [XPATH 1.0] defines the context in which an XPath expression is evaluated.
406 When XPath 1.0 is used as an Expression Language in WS-HumanTask language elements then the
407 XPath context is initialized as follows:

- 408 • Context node: none
- 409 • Context position: none
- 410 • Context size: none
- 411 • Variable bindings: none
- 412 • Function library: Core XPath 1.0 and WS-HumanTask functions MUST be available and
413 processor-specific functions MAY be available
- 414 • Namespace declaration: all in-scope namespace declarations from the enclosing element

415 Note that XPath 1.0 explicitly requires that any element or attribute used in an XPath expression that
416 does not have a namespace prefix must be treated as being namespace unqualified. As a result, even if
417 there is a default namespace defined on the enclosing element, the default namespace will not be
418 applied.

419

3 Concepts

420

3.1 Generic Human Roles

421

Generic human roles define what a person or a group of people resulting from a people query can do with tasks and notifications. The following generic human roles are taken into account in this specification:

423

- Task initiator

424

- Task stakeholders

425

- Potential owners

426

- Actual owner

427

- Excluded owners

428

- Business administrators

429

- Notification recipients

430

431

A *task initiator* is the person who creates the task instance. A WS-HumanTask Definition MAY define assignment for this generic human role. Depending on how the task has been instantiated the task initiator can be defined.

432

433

434

The *task stakeholders* are the people ultimately responsible for the oversight and outcome of the task instance. A task stakeholder can influence the progress of a task, for example, by adding ad-hoc attachments, forwarding the task, or simply observing the state changes of the task. It is also allowed to perform administrative actions on the task instance and associated notification(s), such as resolving missed deadlines. A WS-HumanTask Definition MAY define assignment for this generic human role. WS-HumanTask Processors MUST ensure that at least one person is associated with this role at runtime.

435

436

437

438

439

440

Potential owners of a task are persons who receive the task so that they can claim and complete it. A potential owner becomes the *actual owner* of a task by explicitly claiming it. Before the task has been claimed, potential owners can influence the progress of the task, for example by changing the priority of the task, adding ad-hoc attachments or comments. All excluded owners are implicitly removed from the set of potential owners. A WS-HumanTask Definition MAY define assignment for this generic human role.

441

442

443

444

445

Excluded owners are are people who cannot become an actual or potential owner and thus they cannot reserve or start the task. A WS-HumanTask Definition MAY define assignment for this generic human role.

446

447

448

An *actual owner* of a task is the person actually performing the task. When task is performed, the actual owner can execute actions, such as revoking the claim, forwarding the task, suspending and resuming the task execution or changing the priority of the task. A WS-HumanTask Definition MUST NOT define assignment for this generic human role.

449

450

451

452

Business administrators play the same role as task stakeholders but at task definition level. Therefore, business administrators can perform the exact same operations as task stakeholders. Business administrators can also observe the progress of notifications. A WS-HumanTask Definition MAY define assignment for this generic human role. WS-HumanTask Processors MUST ensure that at runtime at least one person is associated with this role.

453

454

455

456

457

Notification recipients are persons who receive the notification, such as happens when a deadline is missed or when a milestone is reached. This role is similar to the roles potential owners and actual owner but has different repercussions because a notification recipient does not have to perform any action and hence it is more of informational nature than participation. A notification has one or more recipients. A WS-HumanTask Definition MAY define assignment for this generic human role.

458

459

460

461

462 **3.2 Composite Tasks and Sub Tasks**

463 A human task may describe complex work that can be divided into a substructure of related, but
464 independent operations with potential work being carried out by different parties.

465 Complex tasks with substructures are called composite tasks; they can be considered as a composition of
466 multiple (sub) tasks.

467 A sub task describes an act that may or must be completed as part of completing a larger and more
468 complex task. The enclosing composite task may share data with embedded sub tasks, e.g. map data
469 into the input structure of sub tasks or share attachments between composite and sub task.

470 Composite tasks follow the design principle that they are managed by a single task processor.

471 In general sub tasks are human tasks, inheriting all attributes that a human task has, and each behaving
472 the way that a human task does. Some specialties in the area of people assignment and state transitions
473 apply in case a task is a sub task, to align with the behavior of the superior composite task.

474 Tasks can be composite tasks by definition (sub tasks are already defined in the task model) or turn into
475 composite tasks at runtime when a task processor creates in an ad-hoc manner one or more sub tasks to
476 structure work.

477 **3.2.1 Composite Tasks by Definition**

478 In case a composite task is pre-defined as such, the task model contains the definition of one or more sub
479 tasks. Composite tasks come with the following additional attributes:

- 480 • Composition Type (parallel | sequential)
481 Composite tasks with composition type "parallel" allow multiple active sub tasks at the same
482 time; sub tasks are not in any order; composite tasks with composition type "sequential" only
483 allow sequential creation of sub tasks in the pre-defined order (a second listed sub task must not
484 be created before a first listed sub task has been terminated).
- 485 • Creation Pattern (manual | automatic)
486 Composite tasks with activation pattern "manual" expect the "actual owner" to trigger creation of
487 pre-defined sub tasks; composite tasks with activation pattern "automatic" are automatically
488 created at the time the composite task's status becomes "in progress" (where composition type
489 is "parallel" all pre-defined sub tasks are created at the time the composite task's status
490 becomes "in progress"; where composition type is "sequential" at the time the composite task's
491 status becomes "in progress" the first defined sub task will be created; the next sub task in a
492 sequence is automatically created when its predecessor is terminated).

493 **3.2.2 Composite Tasks Created Adhoc at Runtime**

494 An ordinary task may turn into a composite task when the actual owner of a task decides to substructure
495 his work and create sub tasks ad-hoc at runtime.

496 These sub tasks created at runtime behave and are treated as though they are of type "parallel" (a user
497 may create multiple sub tasks at a time) and have an activation pattern of "manual" (creation of ad-hoc
498 sub tasks is always triggered by a user).

499 **3.3 Routing Patterns**

500 A Routing Pattern is a special form of potential owner assignment in which a Task is assigned to people
501 in a well-defined order. Routing patterns allow the assignment of a Task in sequence or parallel. The
502 htd:parallel element defines a parallel routing pattern and the htd:sequence element defines a sequential
503 routing pattern. Those patterns MAY be used in any combination to create complex task routing to
504 people. Routing patterns can be used in both tasks and sub tasks.

505 3.4 Relationship of Composite Tasks and Routing Patterns

506 The complex people assignment used to describe Routing Patterns is a specific syntactic version of
507 Composite Tasks. It is a convenient syntax to describe the "who" in a composite task scenario. The
508 composite task syntax is more expressive to describe the "what" in the sense of which different subtasks
509 are executed.

510 A composite task, including subtasks of different task types, can be described only using the composite
511 task syntax. A routing task containing a dynamic number of subtasks derived from the cardinality of the
512 set of assigned people can be described only using the routing task syntax.

513 Both syntactic flavors may be used in combination which means that a composite task type may include a
514 complex people assignment and that any task defining a complex people assignment may become a
515 composite task at runtime when creating adhoc subtasks.

516 The runtime instantiation model and observable behavior for task instances is identical when using one or
517 the other syntactic flavor.

518 3.5 Assigning People

519 To determine who is responsible for acting on a human task in a certain generic human role or who will
520 receive a notification, people need to be assigned. People assignment can be achieved in different ways:

- 521 • Via logical people groups (see 3.5.1 "Using Logical People Groups")
- 522 • Via literals (see 3.5.2 "Using Literals")
- 523 • Via expressions e.g., by retrieving data from the input message of the human task (see 3.5.3
524 "Using Expressions").
- 525 • In a well-defined order using Routing Patterns (see Routing Patterns)

526 When specifying people assignments then the data type `htd:tOrganizationalEntity` is used. The
527 `htd:tOrganizationalEntity` element specifies the people assignments associated with generic
528 human roles used.

529 Human tasks might be assigned to people in a well-defined order. This includes assignments in a specific
530 sequence and or parallel assignment to a set of people or any combination of both.

531 Syntax:

```
532 <htd:peopleAssignments>  
533  
534   <htd:genericHumanRole>+  
535     <htd:from>...</htd:from>  
536   </htd:genericHumanRole>  
537  
538   <htd:potentialOwners>+  
539     fromPattern+  
540   </htd: potentialOwners>  
541  
542 </htd:peopleAssignments>
```

543 The following syntactical elements for generic human roles are introduced. They can be used wherever
544 the abstract element `genericHumanRole` is allowed by the WS-HumanTask XML Schema.

```
545 <htd:excludedOwners>  
546   <htd:from>...</htd:from>  
547 </htd:excludedOwners>  
548  
549 <htd:taskInitiator>  
550   <htd:from>...</htd:from>  
551 </htd:taskInitiator>
```

```
552
553 <htd:taskStakeholders>
554   <htd:from>...</htd:from>
555 </htd:taskStakeholders>
556
557 <htd:businessAdministrators>
558   <htd:from>...</htd:from>
559 </htd:businessAdministrators>
560
561 <htd:recipients>
562   <htd:from>...</htd:from>
563 </htd:recipients>
```

564 For the potentialOwner generic human role the syntax is as following

```
565 <htd:potentialOwner>
566   fromPattern+
567 </htd:potentialOwner>
568
569 where fromPattern is one of:
570
571 <htd:from> ... </htd:from>
572
573 <htd:sequence type="all|single"?>
574   fromPattern*
575 </htd:sequence>
576
577 <htd:parallel type="all|single"?>
578   fromPattern*
579 </htd:parallel>
```

580 Element <htd:from> is used to specify the value to be assigned to a role. The element has different
581 forms as described below.

582 3.5.1 Using Logical People Groups

583 A *logical people group* represents one person, a set of people, or one or many unresolved groups of
584 people (i.e., group names). A logical people group is bound to a people query against a people directory
585 at deployment time. Though the term *query* is used, the exact discovery and invocation mechanism of this
586 query is not defined by this specification. There are no limitations as to how the logical people group is
587 evaluated. At runtime, this people query is evaluated to retrieve the actual people assigned to the task or
588 notification. Logical people groups **MUST** support query parameters which are passed to the people
589 query at runtime. Parameters **MAY** refer to task instance data (see section 3.8 for more details). During
590 people query execution a WS-HumanTask Processor can decide which of the parameters defined by the
591 logical people group are used. A WS-HumanTask Processor **MAY** use zero or more of the parameters
592 specified. It **MAY** also override certain parameters with values defined during logical people group
593 deployment. The deployment mechanism for tasks and logical people groups is out of scope for this
594 specification.

595 A logical people group has one instance per set of unique arguments. Whenever a logical people group is
596 referenced for the first time with a given set of unique arguments, a new instance **MUST** be created by
597 the WS-HumanTask Processor. To achieve that, the logical people group **MUST** be evaluated / resolved
598 for this set of arguments. Whenever a logical people group is referenced for which an instance already
599 exists (i.e., it has already been referenced with the same set of arguments), the logical people group **MAY**
600 be re-evaluated/re-resolved.

601 In particular, for a logical people group with no parameters, there is a single instance, which MUST be
602 evaluated / resolved when the logical people group is first referenced, and which MAY be re-evaluated /
603 re-resolved when referenced again.

604 People queries are evaluated during the creation of a human task or a notification. If a people query fails
605 a WS-HumanTask Processor MUST create the human task or notification anyway. Failed people queries
606 MUST be treated like people queries that return an empty result set. If the potential owner people query
607 returns an empty set of people a WS-HumanTask Processor MUST perform nomination (see section
608 4.10.1 "Normal processing of a Human Task"). In case of notifications a WS-HumanTask Processor
609 MUST apply the same to notification recipients.

610 People queries return one person, a set of people, or the name of one or many groups of people. The use
611 of a group enables the ability to create a human "work queue" where members are provided access to
612 work items assigned to them as a result of their membership of a group. The ability to defer group
613 membership is beneficial when group membership changes frequently.

614 Logical people groups are global elements enclosed in a human interactions definition document. Multiple
615 human tasks in the same document can utilize the same logical people group definition. During
616 deployment each logical people group is bound to a people query. If two human tasks reference the same
617 logical people group, they are bound to the same people query. However, this does not guarantee that
618 the tasks are actually assigned to the same set of people. The people query is performed for each logical
619 people group reference of a task and can return different results, for example if the content of the people
620 directory has been changed between two queries. Binding of logical people groups to actual people query
621 implementations is out of scope for this specification.

622 **Syntax:**

```
623 <htd:from logicalPeopleGroup="NCName" >  
624   <htd:argument name="NCName" expressionLanguage="anyURI"? >*>  
625     expression  
626   </htd:argument>  
627 </htd:from>
```

628
629 The `logicalPeopleGroup` attribute refers to a `logicalPeopleGroup` definition. The element
630 `<argument>` is used to pass values used in the people query. The `expressionLanguage` attribute
631 specifies the language used in the expression. The attribute is optional. If not specified, the default
632 language as inherited from the closest enclosing element that specifies the attribute MUST be used by
633 WS-HumanTask Processor.

634 **Example:**

```
635 <htd:potentialOwners>  
636   <htd:from logicalPeopleGroup="regionalClerks" >  
637     <htd:argument name="region" >  
638       htd:getInput("part1")/region  
639     </htd:argument>  
640   </htd:from>  
641 </htd:potentialOwners>
```

642 **3.5.2 Using Literals**

643 People assignments can be defined literally by directly specifying the user identifier(s) or the name(s) of
644 groups using either the `htt:tOrganizationalEntity` or `htt:tUser` data type introduced below
645 (see 3.5.4 "Data Type for Organizational Entities").

646

647

648 **Syntax:**

```
649 <htd:from>
650   <htd:literal>
651     ... literal value ...
652   </htd:literal>
653 </htd:from>
```

654 **Example specifying user identifiers:**

```
655 <htd:potentialOwners>
656   <htd:from>
657     <htd:literal>
658       <htt:organizationalEntity>
659         <htt:user>Alan</htt:user>
660         <htt:user>Dieter</htt:user>
661         <htt:user>Frank</htt:user>
662         <htt:user>Gerhard</htt:user>
663         <htt:user>Ivana</htt:user>
664         <htt:user>Karsten</htt:user>
665         <htt:user>Matthias</htt:user>
666         <htt:user>Patrick</htt:user>
667       </htt:organizationalEntity>
668     </htd:literal>
669   </htd:from>
670 </htd:potentialOwners>
```

671 **Example specifying group names:**

```
672 <htd:potentialOwners>
673   <htd:from>
674     <htd:literal>
675       <htt:organizationalEntity>
676         <htt:group>bpel4people_authors</htt:group>
677       </htt:organizationalEntity>
678     </htd:literal>
679   </htd:from>
680 </htd:potentialOwners>
```

681 **3.5.3 Using Expressions**

682 Alternatively people can be assigned using expressions returning either an instance of the
683 `htt:tOrganizationalEntity` data type or the `htt:tUser` data type introduced below (see 3.5.4
684 "Data Type for Organizational Entities").

685 **Syntax:**

```
686 <htd:from expressionLanguage="anyURI"?>
687   expression
688 </htd:from>
```

689

690 The `expressionLanguage` attribute specifies the language used in the expression. The attribute is
691 optional. If not specified, the default language as inherited from the closest enclosing element that
692 specifies the attribute MUST be used by WS-HumanTask Processor.

693

694

695 **Example:**

```
696 <htd:potentialOwners>
697   <htd:from>htd:getInput("part1")/approvers</htd:from>
698 </htd:potentialOwners>
699
700 <htd:businessAdministrators>
701   <htd:from>
702     htd:except( htd:getInput("part1")/admins,
703               htd:getInput("part1")/globaladmins[0] )
704   </htd:from>
705 </htd:businessAdministrators>
```

706 **3.5.4 Data Type for Organizational Entities**

707 The following XML schema definition describes the format of the data that is returned at runtime when
708 evaluating a logical people group. The result can contain either a list of users or a list of groups. The latter
709 is used to defer the resolution of one or more groups of people to a later point, such as when the user
710 accesses a task list.

```
711 <xsd:element name="organizationalEntity" type="tOrganizationalEntity" />
712 <xsd:complexType name="tOrganizationalEntity">
713   <xsd:choice maxOccurs="unbounded">
714     <xsd:element name="user" type="tUser" />
715     <xsd:element name="group" type="tGroup" />
716   </xsd:choice>
717 </xsd:complexType>
718
719 <xsd:element name="user" type="tUser" />
720 <xsd:simpleType name="tUser">
721   <xsd:restriction base="xsd:string" />
722 </xsd:simpleType>
723
724 <xsd:element name="group" type="tGroup" />
725 <xsd:simpleType name="tGroup">
726   <xsd:restriction base="xsd:string" />
727 </xsd:simpleType>
```

728 **3.5.5 Subtasks**

729 Like a task, a sub task has a set of generic human roles. In case people assignment to a sub task's roles
730 is not defined (neither in the sub task's task definition nor on composite task level (using overwrite
731 mechanisms)) the following default assignments apply (especially valid for ad-hoc scenarios):

- 732 • Task initiator
 - 733 a) Activation pattern "manual" → WS-HumanTask Processor MAY assign the actual owner
734 of the composite task
 - 735 b) Activation pattern "automatic" → WS-HumanTask Processor MAY assign the initiator of
736 the composite task
- 737 • Task stakeholders
 - 738 ○ A WS-HumanTask Processor MAY assign the actual owner of the composite task
- 739 • Potential owners
 - 740 ○ No default assignment (usually potential owners will explicitly be defined)
- 741 • Excluded owners

- 742 ○ A WS-HumanTask Processor MUST assign the excluded owners of the composite task
- 743 (This rule applies always, even though the excluded owners of a sub task may be
- 744 enhanced by additional people)
- 745 ● Business administrators
- 746 ○ A WS-HumanTask Processor MAY assign the business administrators of the composite
- 747 task

748 3.6 Task Rendering

749 Humans require a presentation interface to interact with a machine. This specification covers the service
750 interfaces that enable this to be accomplished, and enables this in different constellations of software
751 from different parties. The key elements are the task list client, the task processor and the applications
752 invoked when a task is executed.

753 It is assumed that a single task instance can be rendered by different task list clients so the task engine
754 does not depend on a single dedicated task list client. Similarly it is assumed that one task list client can
755 present tasks from several task engines in one homogenous list and can handle the tasks in a consistent
756 manner. The same is assumed for notifications.

757 A distinction is made between the rendering of the meta-information associated with the task or
758 notification (*task-description UI* and *task list UI*) (see section 4.3 for more details on presentation
759 elements) and the rendering of the task or notification itself (*task-UI*) used for task execution (see section
760 4.4 for more details on task rendering). For example, the task-description UI includes the rendering of a
761 summary list of pending or completed tasks and detailed meta-information such as a deadlines, priority
762 and description about how to perform the task. It is the task list client that deals with this.

763 The task-UI can be rendered by the task list client or delegated to a rendering application invoked by the
764 task list client. The task definition and notification definition can define different rendering information for
765 the task-UI using different rendering methodologies.

766 Versatility of deployment determines which software within a particular constellation performs the
767 presentation rendering.

768 The task-UI can be specified by a rendering method within the task definition or notification definition. The
769 rendering method is identified by a unique name attribute and specifies the type of rendering technology
770 being used. A task or a notification can have more than one such rendering method, e.g. one method for
771 each environment the task or notification is accessed from (e.g. workstation, mobile device).

772 The task-list UI encompasses all information crucial for understanding the importance of and details about
773 a given task or notification (e.g. task priority, subject and description) - typically in a table-like layout.
774 Upon selecting a task, i.e. an entry in case of a table-like layout, the user is given the opportunity to
775 launch the corresponding task-UI. The task-UI has access to the task instance data, and can comprise
776 and manipulate documents other than the task instance. It can be specified by a rendering method within
777 the task description.

778 3.7 Lean Tasks

779 WS-HumanTask enables the creation of task applications with rich renderings, separate input and output
780 messages, and custom business logic in the portType implementation. However, in the spectrum of
781 possible tasks, from enterprise-wide formal processes to department-wide processes to team specific
782 processes to individual, ad-hoc assignments of work, there are scenarios where the task can be defined
783 simply with metadata and the rendering can be left to the WS-HumanTask Processor. An example of this
784 is a simple to-do task, where no form is required beyond the acknowledgement by the actual owner that
785 the work stated in the name, subject, and description of the task is done. A notification doesn't work in
786 this case since it lacks the ability to track whether the work is done or not, and defining a task with a
787 WSDL and portType is beyond the capabilities of those requiring the work done, such as in a team or
788 individual scenario. Therefore, having a way to define the work required of the task in a simpler way
789 enables a greater breadth of scenarios for these smaller scoped types.

790 A Lean Task is a task that has a reduced set of vendor-specific capabilities which results in increased
791 portability and simplicity. The two pieces of the task XML definition that Lean Tasks lack are the ability to
792 define renderings and custom port types. Throughout the specification uses of the word task refers to
793 both types of tasks unless otherwise noted.

794 When used in constellation 4 of WS-BPEL4People, a Lean Task MUST be started through pre-existing
795 interfaces that do not vary in portType or operation per task. The port and operation MUST instead be
796 shipped as part of the installation of the WS-HumanTask Processor (see section 1.4). Therefore, they
797 also lack the ability to define which portType and operation are used to start the task as part of its XML
798 definition. Instead, a Lean Task uses a sub-element that describes the input message (and a symmetrical
799 output message).

800 While a lean task can have one or more renderings explicitly defined, if it defines zero renderings, the
801 schema of the input message and its contained hints for rendering MUST instead be used.

802 All other WS-HumanTask Client to WS-HumanTask Processor interactions behave exactly as before,
803 implying that the processing of a task on a WS-HumanTask Processor for a Lean Task and for a non-
804 Lean Task MUST be indistinguishable from the perspective of a WS-HumanTask Client.

805 **3.8 Task Instance Data**

806 Task instance data falls into three categories:

- 807 • Presentation data – The data is derived from the task definition or the notification definition such
808 as the name, subject or description.
- 809 • Context data - A set of dynamic properties, such as priority, task state, time stamps and values
810 for all generic human roles.
- 811 • Operational data – The data includes the input message, output message, attachments and
812 comments.

813 **3.8.1 Presentation Data**

814 The presentation data is used, for example, when displaying a task or a notification in the task list client.
815 The presentation data has been prepared for display such as by substituting variables. See section 4.3
816 “Presentation Elements” for more details.

817 **3.8.2 Context Data**

818 The task context includes the following:

- 819 • Task state
- 820 • Priority
- 821 • Values for all generic human roles, i.e. potential owners, actual owner and business
822 administrators
- 823 • Time stamps such as start time, completion time, defer expiration time, and expiration time
- 824 • Skipable indicator

825 A WS-HumanTask Processor MAY extend this set of properties available in the task context. For
826 example, the actual owner might start the execution of a task but does not complete it immediately, in
827 which case an intermediate state could be saved in the task context.

828 **3.8.3 Operational Data**

829 The operational data of a task consists of its input data and output data or fault data, as well as any ad-
830 hoc attachments and comments. The operational data of a notification is restricted to its input data.
831 Operational data is accessed using the XPath extension functions and programming interface.

832 3.8.3.1 Ad-hoc Attachments

833 A WS-HumanTask Processor MAY allow arbitrary additional data to be attached to a task. This additional
834 data is referred to as *task ad-hoc attachments*. An ad-hoc attachment is specified by its name, its type
835 and its content and a system-generated attachment identifier.

836 The `contentType` of an attachment can be any valid XML schema type, including `xsd:any`, or any MIME
837 type. The attachment data is assumed to be of that specified content type.

838 The `contentCategory` of an attachment is a URI used to qualify the `contentType`. While `contentType`
839 contains the type of the attachment, the `contentCategory` specifies the type system used when defining
840 the `contentType`. Predefined values for `contentCategory` are

- 841 • "http://www.w3.org/2001/XMLSchema"; if XML Schema types are used for the
842 `contentType`
- 843 • "http://www.iana.org/assignments/media-types/"; if MIME types are used for the
844 `contentType`

845 The set of values is extensible. A WS-HumanTask Processor MUST support the use of XML Schema
846 types and MIME types as content categories, indicated by the predefined URI values shown above.

847 The `accessType` element indicates if the attachment is specified inline or by reference. In the inline case
848 it MUST contain the string constant "inline". In this case the `value` of the `attachment` data type
849 contains the base64 encoded attachment. In case the attachment is referenced it MUST contain the
850 string "URL", indicating that the `value` of the `attachment` data type contains a URL from where the
851 attachment can be retrieved. Other values of the `accessType` element are allowed for extensibility
852 reasons, for example to enable inclusion of attachment content from content management systems.

853 The `attachedTime` element indicates when the attachment is added.

854 The `attachedBy` element indicates who added the attachment. It could be a user, not a group or a list of
855 users or groups.

856 When an ad-hoc attachment is added to a task, the system returns an identifier that is unique among any
857 attachment for the task. It is then possible to retrieve or delete the attachment by the attachment
858 identifier.

859 Attachment Info Data Type

860 The following data type is used to return attachment information on ad-hoc attachments.

```
861 <xsd:element name="attachmentInfo" type="tAttachmentInfo" />  
862 <xsd:complexType name="tAttachmentInfo">  
863   <xsd:sequence>  
864     <xsd:element name="identifier" type="xsd:anyURI" />  
865     <xsd:element name="name" type="xsd:string" />  
866     <xsd:element name="accessType" type="xsd:string" />  
867     <xsd:element name="contentType" type="xsd:string" />  
868     <xsd:element name="contentCategory" type="xsd:anyURI" />  
869     <xsd:element name="attachedTime" type="xsd:dateTime" />  
870     <xsd:element name="attachedBy" type="htt:tUser" />  
871     <xsd:any namespace="##other" processContents="lax"  
872       minOccurs="0" maxOccurs="unbounded" />  
873   </xsd:sequence>  
874 </xsd:complexType>
```

875 Attachment Data Type

876 The following data type is used to return ad-hoc attachments.

```
877 <xsd:element name="attachment" type="tAttachment" />  
878 <xsd:complexType name="tAttachment">  
879   <xsd:sequence>  
880     <xsd:element ref="attachmentInfo" />
```

```
881     <xsd:element name="value" type="xsd:anyType" />
882   </xsd:sequence>
883 </xsd:complexType>
```

884 3.8.3.2 Comments

885 A WS-HumanTask Processor MAY allow tasks to have associated textual notes added by participants of
886 the task. These notes are collectively referred to as *task comments*. Comments are essentially a
887 chronologically ordered list of notes added by various users who worked on the task. A comment has an
888 ID, comment text, the user and timestamp for creation and the user and timestamp of the last
889 modification. Comments are added, modified or deleted individually, but are retrieved as one group.
890 Comments usage is optional in a task.

891 The `addedTime` element indicates when the comment is added.

892 The `addedBy` element indicates who added the comment. It could be a user, not a group or a list of users
893 or groups.

894 The `lastModifiedTime` element indicates when the comment was last modified.

895 The `lastModifiedBy` element indicates who last modified the comment.

896 Comment Data Type

897 The following data type is used to return comments.

```
898 <xsd:element name="comment" type="tComment" />
899 <xsd:complexType name="tComment">
900   <xsd:sequence>
901     <xsd:element name="id" type="xsd:string" />
902     <xsd:element name="addedTime" type="xsd:dateTime" />
903     <xsd:element name="addedBy" type="htt:tUser" />
904     <xsd:element name="lastModifiedTime" type="xsd:dateTime" />
905     <xsd:element name="lastModifiedBy" type="htd:tUser" />
906     <xsd:element name="text" type="xsd:string" />
907     <xsd:any namespace="##other" processContents="lax"
908       minOccurs="0" maxOccurs="unbounded" />
909   </xsd:sequence>
910 </xsd:complexType>
```

911 Comments can be added to a task and retrieved from a task.

912 3.8.4 Data Types for Task Instance Data

913 The following data types are used to represent instance data of a task or a notification. The data type
914 `htt:tTaskAbstract` is used to provide the summary data of a task or a notification that is displayed
915 on a task list. The data type `htt:tTaskDetails` contains the data of a task or a notification, except ad-
916 hoc attachments, comments and presentation description. The data that is not contained in
917 `htt:tTaskDetails` can be retrieved separately using the task API.

918 Contained presentation elements are in a single language (the context determines that language, e.g.,
919 when a task abstract is returned in response to a simple query, the language from the locale of the
920 requestor is used).

921 The elements `startByExists` and `completeByExists` have a value of "true" if the task has at least
922 one start deadline or at least one completion deadline respectively. The actual times (`startByTime` and
923 `completeByTime`) of the individual deadlines can be retrieved using the query operation (see section
924 7.1.3 "Advanced Query Operation").

925 Note that elements that do not apply to notifications are defined as optional.

926 TaskAbstract Data Type

```
927 <xsd:element name="taskAbstract" type="tTaskAbstract" />
```



```

928 <xsd:complexType name="tTaskAbstract">
929   <xsd:sequence>
930     <xsd:element name="id"
931       type="xsd:string" />
932     <xsd:element name="taskType"
933       type="xsd:string" />
934     <xsd:element name="name"
935       type="xsd:QName" />
936     <xsd:element name="status"
937       type="tStatus" />
938     <xsd:element name="priority"
939       type="tPriority" minOccurs="0" />
940     <xsd:element name="createdTime"
941       type="xsd:dateTime" />
942     <xsd:element name="activationTime"
943       type="xsd:dateTime" minOccurs="0" />
944     <xsd:element name="expirationTime"
945       type="xsd:dateTime" minOccurs="0" />
946     <xsd:element name="isSkipable"
947       type="xsd:boolean" minOccurs="0" />
948     <xsd:element name="hasPotentialOwners"
949       type="xsd:boolean" minOccurs="0" />
950     <xsd:element name="startByTimeExists"
951       type="xsd:boolean" minOccurs="0" />
952     <xsd:element name="completeByTimeExists"
953       type="xsd:boolean" minOccurs="0" />
954     <xsd:element name="presentationName"
955       type="tPresentationName" minOccurs="0" />
956     <xsd:element name="presentationSubject"
957       type="tPresentationSubject" minOccurs="0" />
958     <xsd:element name="renderingMethodExists"
959       type="xsd:boolean" />
960     <xsd:element name="hasOutput"
961       type="xsd:boolean" minOccurs="0" />
962     <xsd:element name="hasFault"
963       type="xsd:boolean" minOccurs="0" />
964     <xsd:element name="hasAttachments"
965       type="xsd:boolean" minOccurs="0" />
966     <xsd:element name="hasComments"
967       type="xsd:boolean" minOccurs="0" />
968     <xsd:element name="escalated"
969       type="xsd:boolean" minOccurs="0" />
970     <xsd:element name="outcome"
971       type="xsd:string" minOccurs="0"/>
972     <xsd:element name="parentTaskId"
973       type="xsd:string" minOccurs="0"/>
974     <xsd:element name="hasSubTasks"
975       type="xsd:boolean" minOccurs="0"/>
976     <xsd:any namespace="##other" processContents="lax"
977       minOccurs="0" maxOccurs="unbounded" />
978   </xsd:sequence>
979 </xsd:complexType>

```

980 TaskDetails Data Type

```

981 <xsd:element name="taskDetails" type="tTaskDetails"/>
982 <xsd:complexType name="tTaskDetails">
983   <xsd:sequence>
984     <xsd:element name="id"

```

```

985         type="xsd:string" />
986     <xsd:element name="taskType"
987         type="xsd:string" />
988     <xsd:element name="name"
989         type="xsd:QName" />
990     <xsd:element name="status"
991         type="tStatus" />
992     <xsd:element name="priority"
993         type="htt:tPriority" minOccurs="0" />
994     <xsd:element name="taskInitiator"
995         type="htt:tUser" minOccurs="0" />
996     <xsd:element name="taskStakeholders"
997         type="htt:tOrganizationalEntity" minOccurs="0" />
998     <xsd:element name="potentialOwners"
999         type="htt:tOrganizationalEntity" minOccurs="0" />
1000     <xsd:element name="businessAdministrators"
1001         type="htt:tOrganizationalEntity" minOccurs="0" />
1002     <xsd:element name="actualOwner"
1003         type="htt:tUser" minOccurs="0" />
1004     <xsd:element name="notificationRecipients"
1005         type="htt:tOrganizationalEntity" minOccurs="0" />
1006     <xsd:element name="createdTime"
1007         type="xsd:dateTime" />
1008     <xsd:element name="createdBy"
1009         type="xsd:string" minOccurs="0" />
1010     <xsd:element name="lastModifiedTime"
1011         type="xsd:dateTime" />
1012     <xsd:element name="lastModifiedBy"
1013         type="xsd:string" minOccurs="0" />
1014     <xsd:element name="activationTime"
1015         type="xsd:dateTime" minOccurs="0" />
1016     <xsd:element name="expirationTime"
1017         type="xsd:dateTime" minOccurs="0" />
1018     <xsd:element name="isSkipable"
1019         type="xsd:boolean" minOccurs="0" />
1020     <xsd:element name="hasPotentialOwners"
1021         type="xsd:boolean" minOccurs="0" />
1022     <xsd:element name="startByTimeExists"
1023         type="xsd:boolean" minOccurs="0" />
1024     <xsd:element name="completeByTimeExists"
1025         type="xsd:boolean" minOccurs="0" />
1026     <xsd:element name="presentationName"
1027         type="tPresentationName" minOccurs="0" />
1028     <xsd:element name="presentationSubject"
1029         type="tPresentationSubject" minOccurs="0" />
1030     <xsd:element name="renderingMethodExists"
1031         type="xsd:boolean" />
1032     <xsd:element name="hasOutput"
1033         type="xsd:boolean" minOccurs="0" />
1034     <xsd:element name="hasFault"
1035         type="xsd:boolean" minOccurs="0" />
1036     <xsd:element name="hasAttachments"
1037         type="xsd:boolean" minOccurs="0" />
1038     <xsd:element name="hasComments"
1039         type="xsd:boolean" minOccurs="0" />
1040     <xsd:element name="escalated"
1041         type="xsd:boolean" minOccurs="0" />
1042     <xsd:element name="searchBy"

```



```

1043         type="xsd:string" minOccurs="0"/>
1044     <xsd:element name="outcome"
1045         type="xsd:string" minOccurs="0"/>
1046     <xsd:element name="parentTaskId"
1047         type="xsd:string" minOccurs="0"/>
1048     <xsd:element name="hasSubTasks"
1049         type="xsd:boolean" minOccurs="0"/>
1050     <xsd:any namespace="##other" processContents="lax"
1051         minOccurs="0" maxOccurs="unbounded"/>
1052 </xsd:sequence>
1053 </xsd:complexType>

```

Common Data Types

```

1054 <xsd:simpleType name="tPresentationName">
1055     <xsd:annotation>
1056         <xsd:documentation>length-restricted string</xsd:documentation>
1057     </xsd:annotation>
1058     <xsd:restriction base="xsd:string">
1059         <xsd:maxLength value="64" />
1060         <xsd:whiteSpace value="preserve" />
1061     </xsd:restriction>
1062 </xsd:simpleType>
1063
1064 <xsd:simpleType name="tPresentationSubject">
1065     <xsd:annotation>
1066         <xsd:documentation>length-restricted string</xsd:documentation>
1067     </xsd:annotation>
1068     <xsd:restriction base="xsd:string">
1069         <xsd:maxLength value="254" />
1070         <xsd:whiteSpace value="preserve" />
1071     </xsd:restriction>
1072 </xsd:simpleType>
1073
1074 <xsd:simpleType name="tStatus">
1075     <xsd:restriction base="xsd:string" />
1076 </xsd:simpleType>
1077
1078 <xsd:simpleType name="tPredefinedStatus">
1079     <xsd:annotation>
1080         <xsd:documentation>for documentation only</xsd:documentation>
1081     </xsd:annotation>
1082     <xsd:restriction base="xsd:string">
1083         <xsd:enumeration value="CREATED" />
1084         <xsd:enumeration value="READY" />
1085         <xsd:enumeration value="RESERVED" />
1086         <xsd:enumeration value="IN_PROGRESS" />
1087         <xsd:enumeration value="SUSPENDED" />
1088         <xsd:enumeration value="COMPLETED" />
1089         <xsd:enumeration value="FAILED" />
1090         <xsd:enumeration value="ERROR" />
1091         <xsd:enumeration value="EXITED" />
1092         <xsd:enumeration value="OBSOLETE" />
1093     </xsd:restriction>
1094 </xsd:simpleType>
1095

```

3.8.5 Sub Tasks

To support sub tasks the task instance data gets enhanced by the following (optional) parameters:

- 1098 • sub tasks → A list of task identifiers for each already-created subtask of the task, including
- 1099 both non-terminated and terminated instances
- 1100 → A list of the names of the sub tasks available for creation in the definition of the
- 1101 task, based on the composition type, instantiation pattern, and already created tasks
- 1102 • parent task → The identifier of the superior composite task of this task if it is a sub task

1103 4 Human Tasks

1104 The <task> element is used to specify human tasks. This section introduces the syntax for the element,
1105 and individual properties are explained in subsequent sections.

1106 4.1 Overall Syntax

1107 Definition of human tasks:

```
1108 <htd:task name="NCName" actualOwnerRequired="yes|no"?>
1109
1110   <htd:interface portType="QName" operation="NCName"
1111     responsePortType="QName"? responseOperation="NCName"? />
1112
1113   <htd:priority expressionLanguage="anyURI"? >?
1114     integer-expression
1115   </htd:priority>
1116
1117   <htd:peopleAssignments>?
1118     ...
1119   </htd:peopleAssignments>
1120
1121   <htd:completionBehavior>?
1122     ...
1123   </htd:completionBehavior>
1124
1125   <htd:delegation
1126     potentialDelegates="anybody|nobody|potentialOwners|other" />?
1127     <htd:from>?
1128       ...
1129     </htd:from>
1130   </htd:delegation>
1131
1132   <htd:presentationElements>?
1133     ...
1134   </htd:presentationElements>
1135
1136   <htd:possibleOutcomes>?
1137     ...
1138   </htd:possibleOutcomes>
1139
1140   <htd:outcome part="NCName" queryLanguage="anyURI">?
1141     queryContent
1142   </htd:outcome>
1143
1144   <htd:searchBy expressionLanguage="anyURI"? >?
1145     expression
1146   </htd:searchBy>
1147
1148   <htd:renderings>?
1149     <htd:rendering type="QName">+
1150       ...
1151     </htd:rendering>
1152   </htd:renderings>
1153
1154   <htd:deadlines>?
```

```

1155 <htd:startDeadline name="NCName">*
1156   ...
1157 </htd:startDeadline>
1158
1159 <htd:completionDeadline name="NCName">*
1160   ...
1161 </htd:completionDeadline>
1162
1163 </htd:deadlines>
1164
1165 <htd:composition>?
1166   ...
1167 </htd:composition>
1168
1169
1170 </htd:task>

```

1171 4.2 Properties

1172 The following attributes and elements are defined for tasks:

- 1173 • **name**: This attribute is used to specify the name of the task. The name combined with the target
1174 namespace **MUST** uniquely identify a task element enclosed in the task definition. This attribute
1175 is mandatory. It is not used for task rendering.
- 1176 • **actualOwnerRequired**: This optional attribute specifies if an actual owner is required for the
1177 task. Setting the value to "no" is used for composite tasks where subtasks should be activated
1178 automatically without user interaction. For routing tasks this attribute **MUST** be set to "no". Tasks
1179 that have been defined to not have subtasks **MUST** have exactly one actual owner after they
1180 have been claimed. For these tasks the value of the attribute value **MUST** be "yes". The default
1181 value for the attribute is "yes".
- 1182 • **interface**: This element is used to specify the operation used to invoke the task. The operation
1183 is specified using WSDL, that is, a WSDL port type and WSDL operation are defined. The
1184 element and its `portType` and `operation` attributes **MUST** be present for normal tasks. The
1185 schema only marks it optional so that Lean Tasks can make it prohibited. The interface is
1186 specified in one of the following forms:
 - 1187 ▪ The WSDL operation is a **one-way** operation and the task asynchronously
1188 returns output data. In this case, a WS-HumanTask Definition **MUST** specify a
1189 callback one-way operation, using the `responsePortType` and
1190 `responseOperation` attributes. This callback operation is invoked when the
1191 task has finished. The Web service endpoint address of the callback operation is
1192 provided at runtime when the task's one-way operation is invoked (for details,
1193 see section 10 "Providing Callback Information for Human Tasks").
 - 1194 ▪ The WSDL operation is a **request-response** operation. In this case, the
1195 `responsePortType` and `responseOperation` attributes **MUST NOT** be
1196 specified.
- 1197 • **priority**: This element is used to specify the priority of the task. It is an optional element which
1198 value is an integer expression. If present, the WS-HumanTask Definition **MUST** specify a value
1199 between 0 and 10, where 0 is the highest priority and 10 is the lowest. If not present, the priority
1200 of the task is considered as 5. The result of the expression evaluation is of type
1201 `htt:tPriority`. The `expressionLanguage` attribute specifies the language used in the
1202 expression. The attribute is optional. If not specified, the default language as inherited from the
1203 closest enclosing element that specifies the attribute is used.

- 1204 • `peopleAssignments`: This element is used to specify people assigned to different generic
1205 human roles, i.e. potential owners, and business administrator. The element is optional. See
1206 section 3.5 for more details on people assignments.
 - 1207 • `completionBehavior`: This element is used to specify completion conditions of the task. It is
1208 optional. See section 4.8 for more details on completion behavior.
 - 1209 • `delegation`: This element is used to specify constraints concerning delegation of the task.
1210 Attribute `potentialDelegates` defines to whom the task can be delegated. One of the
1211 following values **MUST** be specified:
 - 1212 ▪ `anybody`: It is allowed to delegate the task to anybody
 - 1213 ▪ `potentialOwners`: It is allowed to delegate the task to potential owners
1214 previously selected
 - 1215 ▪ `other`: It is allowed to delegate the task to other people, e.g. authorized owners.
1216 The element `<from>` is used to determine the people to whom the task can be
1217 delegated.
 - 1218 ▪ `nobody`: It is not allowed to delegate the task.
- 1219 The delegation element is optional. If this element is not present the task is allowed to be
1220 delegated to anybody.
- 1221 • `presentationElements`: This element is used to specify different information used to display
1222 the task in a task list, such as name, subject and description. See section 4.3 for more details on
1223 presentation elements. The element is optional.
 - 1224 • `outcome`: This optional element identifies the field (of an xsd simple type) in the output message
1225 which reflects the business result of the task. A conversion takes place to yield an outcome of
1226 type `xsd:string`. The optional attribute `queryLanguage` specifies the language used for
1227 selection. If not specified, the default language as inherited from the closest enclosing element
1228 that specifies the attribute is used.
 - 1229 • `searchBy`: This optional element is used to search for task instances based on a custom search
1230 criterion. The result of the expression evaluation is of type `xsd:string`. The
1231 `expressionLanguage` attribute specifies the language used in the expression. The attribute is
1232 optional. If not specified, the default language as inherited from the closest enclosing element that
1233 specifies the attribute is used.
 - 1234 • `rendering`: This element is used to specify the rendering method. It is optional. If not present,
1235 task rendering is implementation dependent. See section 4.4 for more details on rendering tasks.
 - 1236 • `deadlines`: This element specifies different deadlines. It is optional. See section 4.9 for more
1237 details on timeouts and escalations.
 - 1238 • `composition`: This element is used to specify subtasks of a composite task. It is optional. See
1239 section 4.6 for more details on composite tasks.

1240 4.3 Presentation Elements

1241 Information about human tasks or notifications needs to be made available in a human-readable way to
1242 allow users dealing with their tasks and notifications via a user interface, which could be based on various
1243 technologies, such as Web browsers, Java clients, Flex-based clients or .NET clients. For example, a
1244 user queries for her tasks, getting a list of tasks she could work on, displaying a short description of each
1245 task. Upon selection of one of the tasks, more complete information about the task is displayed by the
1246 user interface.

1247 Alternatively, a task or notification could be sent directly to a user's inbox, in which case the same
1248 information would be used to provide a human readable rendering there.

1249 The same human readable information could also be used in reports on all the human tasks executed by
1250 a particular human task management system.

1251 Human readable information can be specified in multiple languages.

1252 **Syntax:**

```
1253 <htd:presentationElements>
1254
1255   <htd:name xml:lang="xsd:language"? >*
1256     Text
1257   </htd:name>
1258
1259   <!-- For the subject and description only,
1260     replacement variables can be used. -->
1261   <htd:presentationParameters expressionLanguage="anyURI"? >?
1262     <htd:presentationParameter name="NCName" type="QName">+
1263       expression
1264     </htd:presentationParameter>
1265   </htd:presentationParameters>
1266
1267   <htd:subject xml:lang="xsd:language"? >*
1268     Text
1269   </htd:subject>
1270
1271   <htd:description xml:lang="xsd:language"?
1272     contentType="mimeTypeString"? >*
1273     <xsd:any minOccurs="0" maxOccurs="unbounded" />
1274   </htd:description>
1275
1276 </htd:presentationElements>
```

1277 **Properties**

1278 The following attributes and elements are defined for the `htd:presentationElements` element.

- 1279 • `name`: This element is the short title of a task. It uses `xml:lang`, a standard XML attribute, to
1280 define the language of the enclosed information. This attribute uses tags according to RFC 1766
1281 (see [RFC1766]). There could be zero or more `name` elements. A WS-HumanTask Definition
1282 MUST NOT specify multiple `name` elements having the same value for attribute `xml:lang`.
- 1283 • `presentationParameters`: This element specifies parameters used in presentation elements
1284 `subject` and `description`. Attribute `expressionLanguage` identifies the expression
1285 language used to define parameters. This attribute is optional. If not specified, the default
1286 language as inherited from the closest enclosing element that specifies the attribute is used.
1287 Element `presentationParameters` is optional and if present then the WS-HumanTask
1288 Definition MUST specify at least one element `presentationParameter`. Element
1289 `presentationParameter` has attribute `name`, which uniquely identifies the parameter
1290 definition within the `presentationParameters` element, and attribute `type` which defines its
1291 type. A WS-HumanTask Definition MUST specify parameters of XSD simple types. When a
1292 `presentationParameter` is used within `subject` and `description`, the syntax is
1293 `{$parameterName}`. The pair "`{{`" represents the character "`{`" and the pair "`}}`" represents
1294 the character "`}`". Only the defined presentation parameters are allowed, that is, a WS-
1295 HumanTask Definition MUST NOT specify arbitrary expressions embedded in this syntax.
- 1296 • `subject`: This element is a longer text that describes the task. It uses `xml:lang` to define the
1297 language of the enclosed information. There could be zero or more `subject` elements. A WS-
1298 HumanTask Definition MUST NOT specify multiple `subject` elements having the same value for
1299 attribute `xml:lang`.
- 1300 • `description`: This element is a long description of the task. It uses `xml:lang` to define the
1301 language of the enclosed information. The optional attribute `contentType` uses content types

1302 according to RFC 2046 (see [RFC 2046]). The default value for this attribute is "text/plain". A WS-
1303 HumanTask Processor MUST support the content type "text/plain". The WS-HumanTask
1304 Processor SHOULD support HTML (such as "text/html" or "application/xml+xhtml"). There could
1305 be zero or more description elements. As descriptions can exist with different content types, it
1306 is allowed to specify multiple description elements having the same value for attribute
1307 xml:lang, but the WS-HumanTask Definition MUST specify different content types.

1308 **Example:**

```
1309 <htd:presentationElements>
1310
1311   <htd:name xml:lang="en-US">Approve Claim</htd:name>
1312   <htd:name xml:lang="de-DE">
1313     Genehmigung der Schadensforderung
1314   </htd:name>
1315
1316   <htd:presentationParameters>
1317     <htd:presentationParameter name="firstname" type="xsd:string">
1318       htd:getInput("ClaimApprovalRequest")/cust/firstname
1319     </htd:presentationParameter>
1320     <htd:presentationParameter name="lastname" type="xsd:string">
1321       htd:getInput("ClaimApprovalRequest")/cust/lastname
1322     </htd:presentationParameter>
1323     <htd:presentationParameter name="euroAmount" type="xsd:double">
1324       htd:getInput("ClaimApprovalRequest")/amount
1325     </htd:presentationParameter>
1326   </htd:presentationParameters>
1327
1328   <htd:subject xml:lang="en-US">
1329     Approve the insurance claim for €{$euroAmount} on behalf of
1330     {$firstname} {$lastname}
1331   </htd:subject>
1332   <htd:subject xml:lang="de-DE">
1333     Genehmigung der Schadensforderung über €{$euroAmount} für
1334     {$firstname} {$lastname}
1335   </htd:subject>
1336
1337   <htd:description xml:lang="en-US" contentType="text/plain">
1338     Approve this claim following corporate guideline #4711.0815/7 ...
1339   </htd:description>
1340   <htd:description xml:lang="en-US" contentType="text/html">
1341     <p>
1342       Approve this claim following corporate guideline
1343       <b>#4711.0815/7</b>
1344       ...
1345     </p>
1346   </htd:description>
1347   <htd:description xml:lang="de-DE" contentType="text/plain">
1348     Genehmigen Sie diese Schadensforderung entsprechend Richtlinie Nr.
1349     4711.0815/7 ...
1350   </htd:description>
1351   <htd:description xml:lang="de-DE" contentType="text/html">
1352     <p>
1353       Genehmigen Sie diese Schadensforderung entsprechend Richtlinie
1354       <b>Nr. 4711.0815/7</b>
1355       ...
1356     </p>
1357   </htd:description>
```

1358
1359 `</htd:presentationElements>`

1360 4.4 Task Possible Outcomes

1361 The `<possibleOutcomes>` element provides a way for a task to define which values are usable for the
1362 outcome value of a task. Having a separate definition allows a tool for building tasks to provide support
1363 that understands exactly which outcomes are possible for a particular task.

```
1364 <htd:possibleOutcomes>  
1365   <htd:possibleOutcome name="NCName">+  
1366     <htd:outcomeName xml:lang="xsd:language"?>+  
1367       Language specific display  
1368     </htd:outcomeName>  
1369   </htd:possibleOutcome>  
1370 </htd:possibleOutcomes>
```

1371 Each `<possibleOutcome>` element represents one possible outcome. For the typical example of an
1372 expense report approval, the two outcomes might be 'Approve' and 'Reject'. In addition to the other data
1373 being collected by the rendering in the WS-HumanTask Client, this represents the most important
1374 information about how to proceed in a process that contains multiple tasks. Therefore, a rendering and
1375 client using HTML might choose to show this as a dropdown list, list box with single selection, a set of
1376 submit buttons, or a radio button group.

1377 For each `<possibleOutcome>`, it is possible to have an `<outcomeName>` element to specify a per-
1378 language display name. It uses `xml:lang`, a standard XML attribute, to define the language of the
1379 enclosed information. This attribute uses tags according to RFC 1766 (see [RFC1766]). There could be
1380 zero or more `<outcomeName>` elements. A `<possibleOutcome>` MUST NOT specify multiple
1381 `<outcomeName>` elements having the same value for attribute `xml:lang`.

1382 4.5 Elements for Rendering Tasks

1383 Human tasks and notifications need to be rendered on user interfaces like forms clients, portlets, e-mail
1384 clients, etc. The rendering element provides an extensible mechanism for specifying UI renderings for
1385 human tasks and notifications (task-UI). The element is optional. One or more rendering methods can be
1386 provided in a task definition or a notification definition. A task or notification can be deployed on any WS-
1387 HumanTask Processor, irrespective of the fact whether the implementation supports specified rendering
1388 methods or not. The rendering method is identified using a QName.

1389 Unlike for presentation elements, language considerations are opaque for the rendering element because
1390 the rendering applications typically provide multi-language support. Where this is not the case, providers
1391 of certain rendering types can decide to extend the rendering method in order to provide language
1392 information for a given rendering.

1393 The content of the rendering element is not defined by this specification. For example, when used in the
1394 rendering element, XPath extension functions as defined in section 7.2 MAY be evaluated by a WS-
1395 HumanTask Processor.

1396

1397

1398 **Syntax:**

```
1399 <htd:renderings>
1400   <htd:rendering type="QName">+
1401     <xsd:any minOccurs="1" maxOccurs="1" />
1402   </htd:rendering>
1403 </htd:renderings>
```

1404 4.6 Elements for Composite Tasks

1405 A composite task is defined as a `<htd:task>` element with the `<htd:composition>` element enclosed
1406 in it. The following are attributes and elements defined for the `composition` element.

- 1407 • `type`: This optional attribute specifies the order in which enclosed sub-tasks are executed. If the
1408 value is set to "sequential" the sub-tasks MUST be executed in lexical order. Otherwise they
1409 MUST be executed in parallel. The default value for this attribute is "sequential".
- 1410 • `instantiationPattern`: This optional attribute specifies the way how sub-tasks are
1411 instantiated. If the value is set to "manual" the task client triggers instantiation of enclosed sub-
1412 tasks. Otherwise, they are automatically instantiated at the time the composite task itself turns
1413 into status "inProgress". The default value for this attribute is "manual".
- 1414 • `subtask`: This element specifies a task that will be executed as part of the composite task
1415 execution. The `composition` element MUST enclose at least one `subtask` element. The
1416 `subtask` element has the following attributes and elements. The `name` attribute specifies the
1417 name of the sub-task. The name MUST be unique among the names of all sub-tasks within the
1418 `composition` element. The `htd:task` element is used to define the task inline. The
1419 `htd:localTask` element is used to reference a task that will be executed as a sub-task. The
1420 `htd:localTask` element MAY define values for standard overriding attributes: `priority` and
1421 `people assignments`. The `toParts` element is used to assign values to input message of the
1422 sub-task. The enclosed XPath expression MAY refer to the input message of the composite task
1423 or the output message of other sub-task enclosed in the same `composition` element. The
1424 `part` attribute refers to a part of the WSDL message type of the message used in the XPath.
1425 The `expressionLanguage` attribute specifies the expression language used in the enclosing
1426 elements. The default value for this attribute is `urn:ws-ht:sublang:xpath1.0` which
1427 represents the usage of XPath 1.0 within human interactions definition. A WS-HumanTask
1428 Definition that uses expressions MAY override the default expression language for individual
1429 expressions.

1430 When `composition` is defined on a task, the `composition` MUST be applied for each of the potential
1431 owners defined in the task's `people assignment`.

1432 **Syntax:**

```
1433 <htd:task>
1434   ...
1435   <htd:composition type="sequential|parallel"
1436     instantiationPattern="manual|automatic">
1437     <htd:subtask name="NCName">+
1438
1439     ( <htd:task>
1440       ...
1441     </htd:task>
1442
1443     | <htd:localTask reference="QName">
1444       standard-overriding-elements
1445       ...
1446     </htd:localTask>
```

```

1447     )
1448
1449     <htd:toParts>?
1450     <htd:toPart part="NCName" expressionLanguage="anyURI">+
1451     XPath expression
1452     </htd:toPart>
1453 </htd:toParts>
1454
1455 </htd:subtask>
1456
1457 </htd:composition>
1458 ...
1459 </htd:task>

```

1460 *Standard-overriding-elements* is used in the syntax above as a shortened form of the following list of
 1461 elements:

```

1462 <htd:priority expressionLanguage="anyURI"? >
1463 integer-expression
1464 </htd:priority>
1465
1466 <htd:peopleAssignments>?
1467 <htd:genericHumanRole>
1468 <htd:from>...</htd:from>
1469 </htd:genericHumanRole>
1470 </htd:peopleAssignments>

```

1471 4.7 Elements for People Assignment

1472 The <peopleAssignments> element is used to assign people to a task. For each generic human role, a
 1473 people assignment element can be specified. A WS-HumanTask Definition MUST specify a people
 1474 assignment for potential owners of a human task. An empty <potentialOwners> element is used to
 1475 specify that no potential owner is assigned by the human task's definition but another means is used e.g.
 1476 nomination. Specifying people assignments for task stakeholders, task initiators, excluded owners and
 1477 business administrators is optional. Human tasks never specify recipients. A WS-HumanTask Definition
 1478 MUST NOT specify people assignments for actual owners.

1479 Syntax:

```

1480 <htd:peopleAssignments>
1481
1482 <htd:potentialOwners>
1483 ...
1484 </htd:potentialOwners>
1485
1486 <htd:excludedOwners>?
1487 ...
1488 </htd:excludedOwners>
1489
1490 <htd:taskInitiator>?
1491 ...
1492 </htd:taskInitiator>
1493
1494 <htd:taskStakeholders>?
1495 ...
1496 </htd:taskStakeholders>
1497
1498 <htd:businessAdministrators>?
1499 ...
1500 </htd:businessAdministrators>

```

1501
1502 `</htd:peopleAssignments>`

1503 People assignments can result in a set of values or an empty set. In case people assignment results in an
1504 empty set then the task potentially requires administrative attention. This is out of scope of the
1505 specification, except for people assignments for potential owners (see section 4.10.1 “Normal processing
1506 of a Human Task” for more details).

1507 **Example:**

```
1508 <htd:peopleAssignments>
1509   <htd:potentialOwners>
1510     <htd:from logicalPeopleGroup="regionalClerks">
1511       <htd:argument name="region">
1512         htd:getInput("ClaimApprovalRequest")/region
1513       </htd:argument>
1514     </htd:from>
1515   </htd:potentialOwners>
1516
1517   <htd:businessAdministrators>
1518     <htd:from logicalPeopleGroup="regionalManager">
1519       <htd:argument name="region">
1520         htd:getInput("ClaimApprovalRequest")/region
1521       </htd:argument>
1522     </htd:from>
1523   </htd:businessAdministrators>
1524 </htd:peopleAssignments>
```

1525 **4.7.1 Routing Patterns**

1526 Tasks can be assigned to people in sequence and parallel. Elements `htd:sequence` and
1527 `htd:parallel` elements in `htd:potentialOwners` are used to represent such assignments.

1528 **4.7.1.1 Parallel Pattern**

1529 A task can be assigned to people in parallel using the `htd:parallel` element. . The `htd:parallel`
1530 element is defined as follows:

- 1531 • The `htd:from` element defines the parallel potential owners. This can evaluate to multiple
1532 users/groups.
- 1533 • The attribute ‘type’ in `htd:parallel` identifies how parallel assignments are created for the
1534 multiple users/groups returned from `htd:from`. If type is ‘all’ then an assignment MUST be
1535 created for each user returned by `htd:from`. If type is ‘single’ then an assignment MUST be
1536 created for each `htd:from` clause (this assignment could have with n potential owners). The
1537 default value of type is ‘all’.
- 1538 • The `htd:parallel` and `htd:sequence` elements define nested routing patterns within the
1539 parallel routing pattern
- 1540 • The `htd:completionBehavior` defines when the routing pattern completes. The completion
1541 criteria also define how the result is constructed for the parent task when a parallel routing
1542 pattern is complete.

1543 Each parallel assignment MUST result in a separate sub task. Sub tasks created for each parallel
1544 assignment MUST identify the parent task using the `htd:parentTaskId`.

1545

1546

1547 **Syntax:**

```
1548 <htd:potentialOwners>
1549   <htd:parallel type="all|single"?>
1550     <htd:completionBehavior>
1551       <htd:from>...</htd:from>*
1552       pattern*
1553     </htd:parallel>
1554 </htd:potentialOwners>
```

1555 **Example:**

```
1556 <htd:peopleAssignments>
1557   <htd:potentialOwners>
1558     <htd:parallel type="all">
1559       <htd:from>
1560         htd:getInput("ClaimApprovalRequest")/claimAgent
1561       </htd:from>
1562     </htd:parallel>
1563   </htd:potentialOwners>
1564 </htd:peopleAssignments>
```

1565 4.7.1.2 Sequential Pattern

1566 A task can be assigned to people in sequence using the `htd:sequence` element. The `htd:sequence`
1567 is defined as follows:

- 1568
- The `htd:from` element can evaluate to multiple users/groups.
 - The attribute 'type' in `htd:sequence` identifies how sequential assignments are created for the multiple users/groups returned from `htd:from`. If type is 'all' an assignment MUST be created for each user returned by `htd:from`. If type is 'single', an assignment MUST be created for each `htd:from` clause (this assignment could have with n potential owners). The default value of type is 'all'.
 - The `htd:parallel` and `htd:sequence` elements define nested routing patterns within the sequential routing pattern.
 - The `htd:completionBehavior` defines when the routing pattern completes. The completion criteria also define how the result is constructed for the parent task when a sequential routing pattern is complete.

1579 Sequential routing patterns MUST use a separate sub task for each step in a sequential pattern. Sub
1580 tasks created for each sequential assignment MUST identify the parent task using the
1581 `htd:parentTaskId`.

1582 **Syntax:**

```
1583 <htd:potentialOwners>
1584   <htd:sequence type="all|single"?>
1585     <htd:completionBehavior?>
1586     <htd:from>...</htd:from>*
1587     pattern*
1588   </htd:sequence>
1589 </htd:potentialOwners>
```

1590

1591 **Example:**

```
1592 <htd:peopleAssignments>
1593   <htd:potentialOwners>
1594     <htd:sequence type="all">
1595       <htd:from logicalPeopleGroup="regionalClerks">
1596         <htd:argument name="region">
1597           htd:getInput("ClaimApprovalRequest")/region
1598         </htd:argument>
1599       </htd:from>
1600       <htd:from logicalPeopleGroup="regionalManager">
1601         <htd:argument name="region">
1602           htd:getInput("ClaimApprovalRequest")/region
1603         </htd:argument>
1604       </htd:from>
1605     </htd:sequence>
1606   </htd:potentialOwners>
1607 </htd:peopleAssignments/>
```

1608 **4.8 Completion Behavior**

1609 The completion behavior of a task, routing pattern or composite task can be influenced by a specification
1610 of completion conditions and the result construction for tasks, routing patterns, or composite tasks. For
1611 this purpose, the task, routing pattern or composite task contains a `htd:completionBehavior`
1612 element.

1613 Multiple completion conditions can be specified as nested `htd:completion` elements. They are
1614 evaluated in lexical order. When one of the specified completion conditions is met then the task is
1615 considered to be completed; in case of routing patterns and composite tasks all remaining running sub
1616 tasks **MUST** be skipped (i.e., set to the "Obsolete" state) and the associated result construction **MUST** be
1617 applied.

1618 In case of composite tasks and routing patterns the following applies: At most one default completion
1619 **MUST** be specified with no completion condition in order to specify the result construction after regular
1620 completion of all sub tasks. If no result construction is applied, e.g. because no "default" result
1621 construction is specified and none of the specified completion conditions is met, then the parent task's
1622 output is not created, i.e., it remains uninitialized. Moreover, note that a completion condition can be
1623 specified without referencing sub task output data, which allows the parent task to be considered
1624 completed even without creating any sub tasks. When output data from sub tasks is referenced by
1625 completion conditions or result constructions, only output data of already finished sub tasks **MUST** be
1626 considered.

1627 If none of the specified completion conditions is met then the state of the task or the parent task remains
1628 unchanged.

```
1629 <htd:completionBehavior completionAction="manual|automatic"?>?
1630   <htd:completion name="NCName"> *
1631     <htd:condition ... >
1632       ...
1633     </htd:condition>
1634     <htd:result>?
1635     ...
1636     <htd:result>
1637   </htd:completion>
1638   <htd:defaultCompletion>?
1639     <htd:result>
1640     ...
1641     <htd:result>
1642   </htd:defaultCompletion>
1643 </htd:completionBehavior>
```

1644 The `completionBehavior` element has optional attribute `completionAction`. This optional
1645 attribute specifies how the task, routing pattern, or composite task is completed. If the value is set to
1646 "manual" the task or parent task MUST be completed explicitly by the actual owner as soon as the
1647 completion conditions evaluate to true. If the value is set to "automatic" the task or parent task MUST be
1648 set to complete as soon as the completion conditions evaluate to true. For routing patterns, the
1649 `completionAction` attribute MUST have value "automatic". The default value for this attribute is
1650 "automatic".

1651 If `completionBehavior` is not specified, the default behavior is that of a `completionBehavior` with
1652 `completionCondition` is "true" and a `completionAction` of "manual" for simple and composite
1653 tasks, and "automatic" for routing patterns.

1654 4.8.1 Completion Conditions

1655 A completion condition defines when a task or a set of sub tasks associated with the parent task is
1656 considered completed. It is specified Boolean expression which can refer to input data of the task, the
1657 parent task or its sub tasks, output data produced by already finished sub tasks, or other data obtained
1658 from WS-HumanTask API calls (e.g. the number of sub tasks), or functions that test that some designated
1659 amount of time has passed.

1660 The completion condition MUST be defined using an `htd:condition` element.

```
1661 <htd:condition expressionLanguage="anyURI"?>  
1662   boolean expression  
1663 </htd:condition>
```

1664 Within the Boolean expression of a completion condition, aggregation functions can be used to evaluate
1665 output data produced by the already finished sub tasks of the parent task.

1666 If an error (e.g. division by zero) occurs during the condition evaluation then the condition MUST be
1667 considered to have evaluated to "false".

1668 The time functions that are available are defined as follows:

- 1669 • `boolean htd:waitFor(string)`
 - 1670 ○ The parameter is an XPath expression evaluating to a string conforming to the definition
1671 ○ of the XML Schema type `duration`
 - 1672 ○ The return value is `true` after the specified duration has elapsed, otherwise `false`
- 1673 • `boolean htd:waitUntil(string)`
 - 1674 ○ The parameter is an XPath expression evaluating to a string conforming to the definition
1675 ○ of the XML Schema type `dateTime`

1676 The return value is `true` after the specified absolute time has passed, otherwise `false`.

1677 Completion conditions of a task MUST use only time functions.

1678 4.8.1.1 Evaluating the Completion Condition

1679 The time functions in the completion condition are be evaluated with respect to the beginning of execution
1680 of the task or parent task on which the completion is defined. To achieve this, the evaluation of the
1681 `htd:waitFor` and `htd:waitUntil` calls within the condition are treated differently from the rest of the
1682 expression. When the containing task or parent task is created, the actual parameter expression for any
1683 `htd:waitFor` and `htd:waitUntil` calls MUST be evaluated and the completion condition should be
1684 rewritten to replace these calls with only `htd:waitUntil` calls with constant parameters. The durations
1685 calculated for any `htd:waitFor` calls MUST be converted into absolute times and rewritten as
1686 `htd:waitUntil` calls. The result of these replacements is called the *preprocessed completion*
1687 *condition*.

1688

1689

1690 For the parent task, the preprocessed completion condition MUST be evaluated at the following times:

- 1691 • Before starting the first subtask (it may be complete before it starts)
- 1692 • Whenever a subtask completes
- 1693 • Whenever a duration specified in a `htd:waitFor` call has elapsed
- 1694 • Whenever an absolute time specified in a `htd:waitUntil` call is passed.

1695 For tasks, the preprocessed completion condition MUST be evaluated at the following times:

- 1696 • Before starting the task (it may be complete before it starts)
- 1697 • Whenever a duration specified in a `htd:waitFor` call has elapsed
- 1698 • Whenever an absolute time specified in a `htd:waitUntil` call is passed.

1699 **Example:**

1700 The first completion condition may be met even without starting sub tasks. When both parts of the second
1701 completion condition are met, that is, 7 days have expired and more than half of the finished sub tasks
1702 have an outcome of "Rejected", then the parallel routing pattern is considered completed.

```
1703 <htd:parallel>  
1704   ...  
1705   <htd:completionBehavior>  
1706     <htd:completion>  
1707       <htd:condition>  
1708         htd:getInput("ClaimApprovalRequest")/amount < 1000  
1709       </htd:condition>  
1710       <htd:result> ... </htd:result>  
1711     </htd:completion>  
1712     <htd:completion>  
1713       <htd:condition>  
1714         ( htd:getCountOfSubtasksWithOutcome("Rejected") /  
1715           htd:getCountOfSubtasks() > 0.5 )  
1716         and htd:waitFor("P7D")  
1717       </htd:condition>  
1718       <htd:result> ... </htd:result>  
1719     </htd:completion>  
1720   </htd:completionBehavior>  
1721   ...  
1722 </htd:parallel>
```

1723 **4.8.2 Result Construction from Parallel Subtasks**

1724 When multiple sub tasks are created in order let several people work on their own sub task in parallel
1725 then the outputs of these sub tasks sometimes need to be combined for the creation of the parent task's
1726 output.

1727 If all sub tasks have the same interface definition (as in routing patterns) then the result construction can
1728 be defined in a declarative way using aggregation functions. Alternatively, the result may be created using
1729 explicit assignments.

1730 The result construction MUST be defined as `htd:result` element, containing one or more
1731 `htd:aggregate` or `htd:copy` elements, executed in the order in which they appear in the task
1732 definition.

```
1733 <htd:result>  
1734   (  
1735     <htd:aggregate ... />  
1736     |  
1737     <htd:copy> ... </htd:copy>
```



```
1738 )+
1739 </htd:result>
```

1740 4.8.2.1 Declarative Result Aggregation

1741 An `htd:aggregate` element describes the result aggregation for a leaf element of the parent task's
1742 output document. In most cases, this approach is only meaningful for routing patterns with identical sub
1743 task interfaces. Note that the construction of (complex-typed) non-leaf elements is out of scope of the
1744 declarative result aggregation.

```
1745 <htd:aggregate part="NCName" ?
1746             location="query" ?
1747             condition="bool-expr" ?
1748             function="function-call" />+
```

1749 The `htd:aggregate` element is defined as follows:

- 1750 • The optional `part` attribute MUST contain the name of a WSDL part. The `part` attribute MUST be
1751 specified when the task interface is defined using a WSDL message with more than one WSDL
1752 part.
- 1753 • The optional `location` attribute MUST contain a query pointing to the location of a leaf element
1754 of the tasks' output documents:
 - 1755 ○ For each parallel sub task, this is the location of exactly one element of the sub task's
1756 output document that is processed by the aggregation function. Each sub tasks' output
1757 element is (conditionally) added to a node-set passed as parameter to the aggregation
1758 function.
 - 1759 ○ For the parent task, this is the element created in the task's output document that is the
1760 computed return value of the aggregation function.
- 1761 • The optional `condition` attribute MUST contain a Boolean expression evaluated on each sub
1762 task's output document. If the expression evaluates to `true` then the sub task's output element
1763 identified by `location` is added to the node-set passed to the aggregation function.
- 1764 • The mandatory `function` attribute contains the name of the aggregation function (QName; see
1765 a list of supported aggregation functions below) and optional arguments, in the following form:
1766 `FunctionName '(' (Argument (',' Argument)*)? ')'`
1767 Important:
 - 1768 ○ The first parameter of each aggregation function is the node-set of sub task's output
1769 elements to be aggregated. This parameter is inserted implicitly and MUST NOT be
1770 specified within the `function` attribute.
 - 1771 ○ Within the `function` attribute, function arguments MUST be specified only for *additional*
1772 parameters defined for an aggregation function.

1773 Example:

1774 Consider the following output document used in a parallel routing pattern:

```
1775 <element name="Award" type="tns:tAward" />
1776 <complexType name="tAward">
1777   <sequence>
1778     <element name="AwardRecommended" type="xsd:string" />
1779     <element name="AwardDetails" type="tns:tAwardDetails" />
1780   </sequence>
1781 </complexType>
1782 <complexType name="tAwardDetails">
1783   <sequence>
1784     <element name="Amount" type="xsd:integer" />
1785     <element name="Appraisal" type="xsd:string" />
1786   </sequence>
```

1787 </complexType>

1788 A possible result aggregation could then look like this. The first aggregation determines the most frequent
1789 occurrence of an award recommendation. The second aggregation calculates the average award amount
1790 for sub tasks with an award recommendation of 'yes'. The third aggregation creates a comma-separated
1791 concatenation of all sub task's appraisals.

```
1792 <htd:parallel ...>
1793   ...
1794   <htd:completionBehavior>
1795     <htd:completion>
1796       <htd:condition> ... </htd:condition>
1797       <htd:result>
1798         <htd:aggregate location="/Award/AwardRecommended"
1799           function="htd:mostFrequentOccurence()" />
1800         <htd:aggregate location="/Award/AwardDetails/Amount"
1801           condition="/Award/AwardRecommended='yes'"
1802           function="htd:avg()" />
1803         <htd:aggregate location="/Award/AwardDetails/Appraisal"
1804           function="htd:concatWithDelimiter(',')" />
1805       </htd:result>
1806     </htd:completion>
1807   </htd:completionBehavior>
1808 </htd:parallel>
```

1809 4.8.2.2 Explicit Result Assignment

1810 An `htd:copy` element describes the explicit assignment to an element of the parent task's output
1811 document.

```
1812 <htd:copy>+
1813   <htd:from expressionLanguage="anyURI"?>
1814     expression
1815   </htd:from>
1816   <htd:to queryLanguage="anyURI"?>
1817     query
1818   </htd:to>
1819 </htd:copy>
```

1820 The `htd:copy` element is defined as follows:

- 1821 • The mandatory `htd:from` element MUST contain an expression used to calculate the result
1822 value. The expression can make use of WS-HumanTask aggregation functions.
- 1823 • The mandatory `htd:to` element MUST contain a query pointing to the location of an element of
1824 the tasks' output documents. This is the element created in the task's output document.

1825 Example 1:

1826 Consider the following output document used in a parallel routing pattern:

```
1827 <element name="Order" type="tns:tOrder" />
1828 <complexType name="tOrder">
1829   <sequence>
1830     <element name="Item" type="tns:tItem" maxOccurs="unbounded" />
1831     <element name="TotalPrice" type="xsd:integer" />
1832   </sequence>
1833 </complexType>
1834 <complexType name="tItem">
1835   <sequence>
1836     ...
1837   </sequence>
1838 </complexType>
```

1839 A possible result aggregation could then look like this. All sub task order item lists are concatenated to
1840 one parent task order item list. The total price is calculated using an aggregation function.

```
1841 <htd:parallel>  
1842   ...  
1843   <htd:completionBehavior>  
1844     <htd:completion>  
1845       <htd:condition> ... </htd:condition>  
1846       <htd:result>  
1847         <htd:copy>  
1848           <htd:from>  
1849             htd:getSubtaskOutputs("orderResponse", "/Order/Item")  
1850           </htd:from>  
1851           <htd:to>/Order/Item</htd:to>  
1852         </htd:copy>  
1853         <htd:copy>  
1854           <htd:from>  
1855             htd:sum(htd:getSubtaskOutputs("orderResponse",  
1856               "/Order/TotalPrice"))  
1857           </htd:from>  
1858           <htd:to>/Order/TotalPrice</htd:to>  
1859         </htd:copy>  
1860       </htd:result>  
1861     </htd:completion>  
1862   </htd:completionBehavior>  
1863 </htd:parallel>
```

1864 **Example 2:**

1865 Output data from heterogeneous sub tasks is assigned into the parent task's output. The complete
1866 complex-typed sub task output documents are copied into child elements of the parent task output
1867 document.

```
1868 <htd:task name="bookTrip">  
1869   ... produces itinerary ...  
1870  
1871   <htd:composition type="parallel" ...>  
1872     <htd:subtask name="bookHotel">  
1873       <htd:task>  
1874         ... produces hotelReservation ...  
1875       </htd:task>  
1876     </htd:subtask>  
1877     <htd:subtask name="bookFlight">  
1878       <htd:task>  
1879         ... produces flightReservation ...  
1880       </htd:task>  
1881     </htd:subtask>  
1882   </htd:composition>  
1883   ...  
1884   <htd:completionBehavior>  
1885     <htd:defaultCompletion>  
1886       <htd:result>  
1887         <htd:copy>  
1888           <htd:from>  
1889             htd:getSubtaskOutput("bookHotel",  
1890               "bookHotelResponse",  
1891               "/hotelReservation")  
1892           </htd:from>  
1893           <htd:to>/itinerary/hotelReservation</htd:to>  
1894         </htd:copy>
```

```

1895     <htd:copy>
1896       <htd:from>
1897         htd:getSubtaskOutput("bookFlight",
1898                               "bookFlightResponse",
1899                               "/flightReservation")
1900       </htd:from>
1901     <htd:to>/itinerary/flightReservation</htd:to>
1902   </htd:copy>
1903 </htd:result>
1904 </htd:defaultCompletion>
1905 </htd:completionBehavior>
1906 </htd:task>

```

1907 4.9 Elements for Handling Timeouts and Escalations

1908 Timeouts and escalations allow the specification of a date or time before which the task or sub task has to
 1909 reach a specific state. If the timeout occurs a set of actions is performed as the response. The state of the
 1910 task is not changed. Several deadlines are specified which differ in the point when the timer clock starts
 1911 and the state which has to be reached with the given duration or by the given date. They are:

- 1912 • Start deadline: Specifies the time until the task has to start, i.e. it has to reach state *InProgress*. It
 1913 is defined as either the period of time or the point in time until the task has to reach state
 1914 *InProgress*. Since expressions are allowed, durations and deadlines can be calculated at runtime,
 1915 which for example enables custom calendar integration. The time starts to be measured from the
 1916 time at which the task enters the state *Created*. If the task does not reach state *InProgress* by the
 1917 deadline an escalation action or a set of escalation actions is performed. Once the task is started,
 1918 the timer becomes obsolete.
- 1919 • Completion deadline: Specifies the due time of the task. It is defined as either the period of time
 1920 until the task gets due or the point in time when the task gets due. The time starts to be measured
 1921 from the time at which the task enters the state *Created*. If the task does not reach one of the final
 1922 states (*Completed*, *Failed*, *Error*, *Exited*, *Obsolete*) by the deadline an escalation action or a set
 1923 of escalation actions is performed.

1924 The element <deadlines> is used to include the definition of all deadlines within the task definition. It is
 1925 optional. If present then the WS-HumanTask Definition MUST specify at least one deadline. Deadlines
 1926 defined in ad-hoc sub tasks created at runtime MUST NOT contradict the deadlines of their parent task.
 1927 The value of the name attribute MUST be unique for all deadline specifications within a task definition.

1928 Syntax:

```

1929 <htd:deadlines>
1930
1931   <htd:startDeadline name="NCName" > *
1932
1933     <htd:documentation xml:lang="xsd:language" ? > *
1934       text
1935     </htd:documentation>
1936
1937     ( <htd:for expressionLanguage="anyURI" ? >
1938       duration-expression
1939     </htd:for>
1940     | <htd:until expressionLanguage="anyURI" ? >
1941       deadline-expression
1942     </htd:until>
1943     )
1944
1945   <htd:escalation name="NCName" > *
1946   ...

```

```
1947     </htd:escalation>
1948
1949     </htd:startDeadline>
1950
1951     <htd:completionDeadline name="NCName">*
1952     ...
1953     </htd:completionDeadline>
1954
1955 </htd:deadlines>
```

1956 The language used in expressions is specified using the `expressionLanguage` attribute. This attribute
1957 is optional. If not specified, the default language as inherited from the closest enclosing element that
1958 specifies the attribute is used.

1959 For all deadlines if a status is not reached within a certain time then an escalation action, specified using
1960 element `<escalation>`, can be triggered. The `<escalation>` element is defined in the section below.
1961 When the task reaches a final state (*Completed, Failed, Error, Exited, Obsolete*) all deadlines are deleted.

1962 Escalations are triggered if

- 1963 1. The associated point in time is reached, or duration has elapsed, and
- 1964 2. The associated condition (if any) evaluates to true

1965 Escalations use notifications to inform people about the status of the task. Optionally, a task might be
1966 reassigned to some other person or group as part of the escalation. Notifications are explained in more
1967 detail in section 6 “Notifications”. For an escalation, a WS-HumanTask Definition MUST specify exactly
1968 one escalation action.

1969 When defining escalations, a notification can be either referred to, or defined inline.

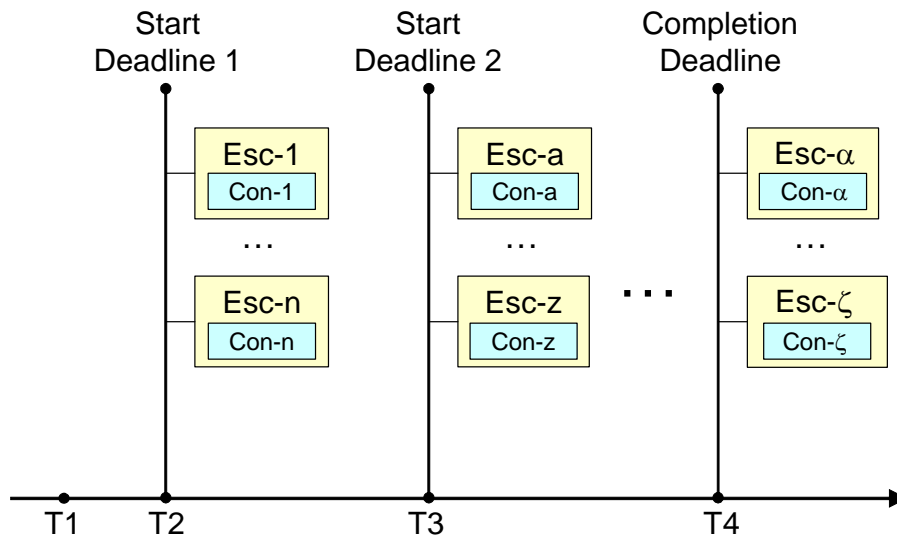
- 1970 • A notification defined in the `<humanInteractions>` root element or imported from a different
1971 namespace can be referenced by specifying its QName in the `reference` attribute of a
1972 `<localNotification>` element. When referring to a notification, the priority and the people
1973 assignments of the original notification definition MAY be overridden using the elements
1974 `<priority>` and `<peopleAssignments>` contained in the `<localNotification>` element.
- 1975 • An inlined notification is defined by a `<notification>` element.

1976 Notifications used in escalations can use the same type of input data as the surrounding task or sub task,
1977 or different type of data. If the same type of data is used then the input message of the task or sub task is
1978 passed to the notification implicitly. If not, then the `<toPart>` elements are used to assign appropriate
1979 data to the notification, i.e. to explicitly create a multi-part WSDL message from the data. The `part`
1980 attribute refers to a part of the WSDL message. The `expressionLanguage` attribute specifies the
1981 language used in the expression. The attribute is optional. If not specified, the default language as
1982 inherited from the closest enclosing element that specifies the attribute is used.

1983 A WS-HumanTask Definition MUST specify a `<toPart>` element for every part in the WSDL message
1984 definition because parts not explicitly represented by `<toPart>` elements would result in uninitialized parts
1985 in the target WSDL message. The order in which parts are specified is not relevant. If multiple `<toPart>`
1986 elements are present, a WS-HumanTask Processor MUST execute them in an “all or nothing” manner. If
1987 any of the `<toPart>`s fails, the escalation action will not be performed and the execution of the task is not
1988 affected.

1989 Reassignments are used to replace the potential owners of a task when an escalation is triggered. The
1990 `<reassignment>` element is used to specify reassignment. If present then a WS-HumanTask Definition
1991 MUST specify potential owners. A reassignment triggered by a sub task escalation MUST apply to the
1992 sub task only. A reassignment MAY comprise of a complex people assignment using Routing Patterns.

1993 In the case where several reassignment escalations are triggered, the first reassignment (lexical order)
1994 MUST be considered for execution by the WS-HumanTask Processor. The task is set to state *Ready* after
1995 reassignment. Reassignments and notifications are performed in the lexical order.



1996

1997 A task MAY have multiple start deadlines and completion deadlines associated with it. Each such
 1998 deadline encompasses escalation actions each of which MAY send notifications to certain people. The
 1999 corresponding set of people MAY overlap.

2000 As an example, the figure depicts a task that has been created at time T1. Its two start deadlines would
 2001 be missed at time T2 and T3, respectively. The associated escalations whose conditions evaluate to
 2002 "true" are triggered. Both, the escalations Esc-1 to Esc-n as well as escalations Esc-a to Esc-z can
 2003 involve an overlapping set of people. The completion deadline would be missed at time T4.

2004 **Syntax:**

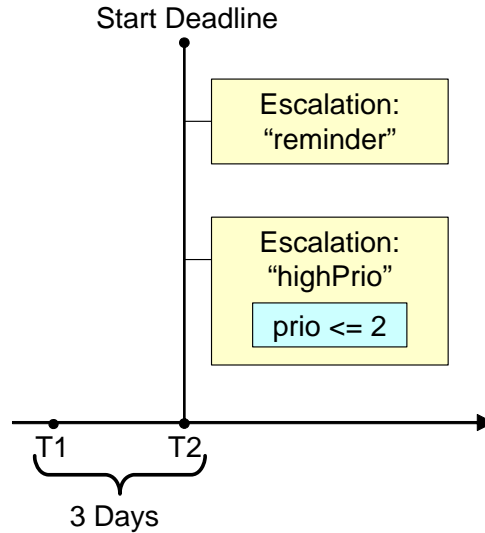
```

2005 <htd:deadlines>
2006
2007   <htd:startDeadline name="NCName" > *
2008     ...
2009     <htd:escalation name="NCName" > *
2010
2011       <htd:condition expressionLanguage="anyURI" ?> ?
2012         boolean-expression
2013       </htd:condition>
2014
2015       <htd:toParts > ?
2016         <htd:toPart part="NCName"
2017           expressionLanguage="anyURI" ?> +
2018           expression
2019         </htd:toPart >
2020       </htd:toParts >
2021
2022       <!-- notification specified by reference -->
2023       <htd:localNotification reference="QName" ?>
2024         <htd:priority expressionLanguage="anyURI" ?> ?
2025         integer-expression
2026       </htd:priority >
2027       <htd:peopleAssignments > ?
2028         <htd:recipients >
2029           ...
2030         </htd:recipients >
2031       </htd:peopleAssignments >
  
```

```
2032
2033     </htd:localNotification>
2034
2035     <!-- notification specified inline -->
2036     <htd:notification name="NCName">?
2037         ...
2038     </htd:notification>
2039
2040     <htd:reassignment>?
2041         <htd:potentialOwners>
2042             ...
2043         </htd:potentialOwners>
2044
2045     </htd:reassignment>
2046
2047 </htd:escalation>
2048
2049 </htd:startDeadline>
2050
2051 <htd:completionDeadline name="NCName">*
2052     ...
2053 </htd:completionDeadline>
2054
2055 </htd:deadlines>
2056
2057
```


2058 **Example:**

2059 The following example shows the specification of a start deadline with escalations. At runtime, the



2060 following picture depicts the result of what is specified in the example:

2061 The human task is created at T1. If it has not been started, i.e., no person is working on it until T2, then
2062 the escalation "reminder" is triggered that notifies the potential owners of the task that work is waiting for
2063 them. In case the task has high priority then at the same time the regional manager is informed. If the
2064 task amount is greater than or equal 10000 the task is reassigned to Alan.

2065 In case that task has been started before T2 was reached, then the start deadline is deactivated, no
2066 escalation occurs.

```
2067 <htd:startDeadline name="sendNotifications">
2068   <htd:documentation xml:lang="en-US">
2069     If not started within 3 days, - escalation notifications are sent
2070     if the claimed amount is less than 10000 - to the task's potential
2071     owners to remind them or their todo - to the regional manager, if
2072     this approval is of high priority (0,1, or 2) - the task is
2073     reassigned to Alan if the claimed amount is greater than or equal
2074     10000
2075   </htd:documentation>
2076   <htd:for>P3D</htd:for>
2077   <htd:escalation name="reminder">
2078
2079     <htd:condition>
2080       <![CDATA[
2081         htd:getInput("ClaimApprovalRequest")/amount < 10000
2082       ]]>
2083     </htd:condition>
2084
2085     <htd:toParts>
2086       <htd:toPart name="firstname">
2087         htd:getInput("ClaimApprovalRequest", "ApproveClaim")/firstname
2088       </htd:toPart>
2089       <htd:toPart name="lastname">
2090         htd:getInput("ClaimApprovalRequest", "ApproveClaim")/lastname
2091       </htd:toPart>
2092     </htd:toParts>
```

```

2093
2094 <htd:localNotification reference="tns:ClaimApprovalReminder">
2095
2096 <htd:documentation xml:lang="en-US">
2097     Reuse the predefined notification "ClaimApprovalReminder".
2098     Overwrite the recipients with the task's potential owners.
2099 </htd:documentation>
2100
2101 <htd:peopleAssignments>
2102 <htd:recipients>
2103     <htd:from>htd:getPotentialOwners("ApproveClaim")</htd:from>
2104 </htd:recipients>
2105 </htd:peopleAssignments>
2106
2107 </htd:localNotification>
2108
2109 </htd:escalation>
2110
2111 <htd:escalation name="highPrio">
2112
2113 <htd:condition>
2114 <![CDATA[
2115         (htd:getInput("ClaimApprovalRequest")/amount < 10000
2116         && htd:getInput("ClaimApprovalRequest")/prio <= 2)
2117     ]]>
2118 </htd:condition>
2119
2120 <!-- task input implicitly passed to the notification -->
2121
2122 <htd:notification name="ClaimApprovalOverdue">
2123 <htd:documentation xml:lang="en-US">
2124     An inline defined notification using the approval data as its
2125     input.
2126 </htd:documentation>
2127
2128 <htd:interface portType="tns:ClaimsHandlingPT"
2129     operation="escalate" />
2130
2131 <htd:peopleAssignments>
2132 <htd:recipients>
2133     <htd:from logicalPeopleGroup="regionalManager">
2134         <htd:argument name="region">
2135             htd:getInput("ClaimApprovalRequest")/region
2136         </htd:argument>
2137     </htd:from>
2138 </htd:recipients>
2139 </htd:peopleAssignments>
2140
2141 <htd:presentationElements>
2142 <htd:name xml:lang="en-US">Claim approval overdue</htd:name>
2143 <htd:name xml:lang="de-DE">
2144     Überfällige Schadensforderungsgenehmigung
2145 </htd:name>
2146 </htd:presentationElements>
2147
2148 </htd:notification>
2149
2150 </htd:escalation>

```

```

2151 <htd:escalation name="highAmountReassign">
2152
2153   <htd:condition>
2154     <![CDATA[
2155       htd:getInput("ClaimApprovalRequest")/amount >= 10000
2156     ]]>
2157   </htd:condition>
2158
2159   <htd:reassignment>
2160     <htd:documentation>
2161       Reassign task to Alan if amount is greater than or equal
2162       10000.
2163     </htd:documentation>
2164
2165     <htd:potentialOwners>
2166       <htd:from>
2167         <htd:literal>
2168           <htt:organizationalEntity>
2169             <htt:user>Alan</htt:user>
2170           </htt:organizationalEntity>
2171         </htd:literal>
2172       </htd:from>
2173     </htd:potentialOwners>
2174
2175   </htd:reassignment>
2176 </htd:escalation>
2177
2178 </htd:startDeadline>
2179
2180

```

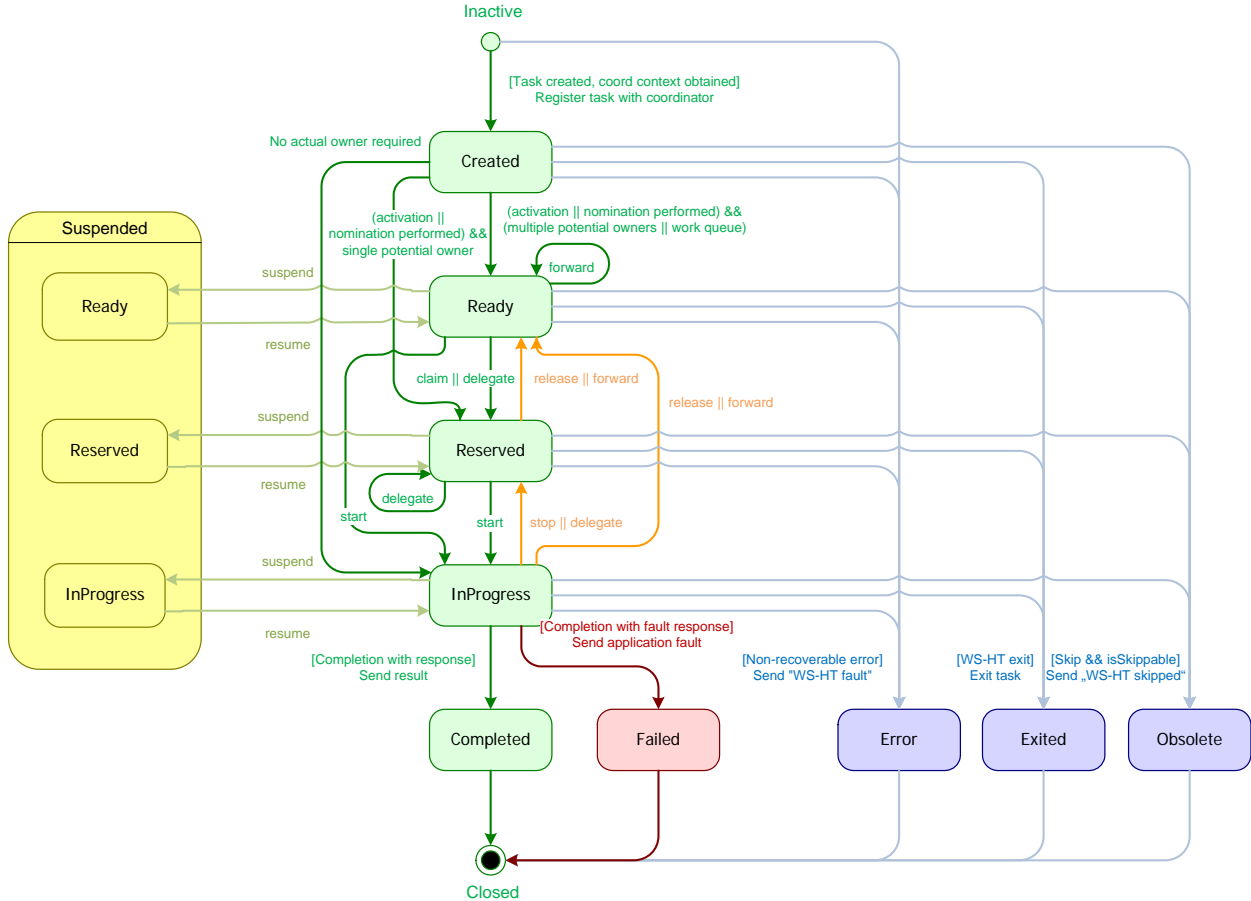
2181 All timeouts and escalations apply to sub tasks also. If htd:escalation is triggered for a sub task, then any
2182 htd:reassignment MUST be applied only to that.

2183

2184

2185 4.10 Human Task Behavior and State Transitions

2186 Human tasks can have a number of different states and substates. The state diagram for human tasks
2187 below shows the different states and the transitions between them.



2188

2189 4.10.1 Normal processing of a Human Task

2190 Upon creation, a task goes into its initial state *Created*. Task creation starts with the initialization of its
2191 properties in the following order:

- 2192 1. Input message
- 2193 2. Priority
- 2194 3. Generic human roles (such as excluded owners, potential owners and business administrators)
2195 are made available in the lexical order of their definition in the people assignment definition with
2196 the constraint that excluded owners are taken into account when evaluating the potential owners.
- 2197 4. All other properties are evaluated after these properties in an implementation dependent order.

2198 Task creation succeeds irrespective of whether the people assignment returns a set of values or an
2199 empty set. People queries that cannot be executed successfully are treated as if they were returning an
2200 empty set.

2201 If potential owners were not assigned automatically during task creation then they **MUST** be assigned
2202 explicitly using nomination, which is performed by the task's business administrator. The result of
2203 evaluating potential owners removes the excluded owners from results. The task remains in the state
2204 *Created* until it is activated (i.e., an activation timer has been specified) and has potential owners.

2205 When the task has a single potential owner, it transitions into the *Reserved* state, indicating that it is
2206 assigned to a single actual owner. Otherwise (i.e., when it has multiple potential owners or is assigned to
2207 a work queue), it transitions into the *Ready* state, indicating that it can be claimed by one of its potential
2208 owners. Once a potential owner claims the task, it transitions into the *Reserved* state, making that
2209 potential owner the actual owner.

2210 Once work is started on a task that is in state *Ready* or *Reserved*, it goes into the *InProgress* state,
2211 indicating that it is being worked on – if the transition is from *Ready*, the user starting the work becomes
2212 its actual owner.

2213 On successful completion of the work, the task transitions into the *Completed* final state. On unsuccessful
2214 completion of the work (i.e., with an exception), the task transitions into the *Failed* final state.

2215 The lifecycle of sub tasks is the same as that of the main task.

2216 For human tasks that have subtasks two different cases exist, with different implications:

2217 1. Tasks with subtasks where an actual owner is required

2218 2. Tasks with subtasks where no actual owner is required

2219 The first case has the sub-case where a potential owner has been modeled on the primary task and
2220 subtasks have been modeled that are activated either manually or automatically. Another sub-case of the
2221 first case is the one where no potential owner has been modeled and thus nomination has to occur. In all
2222 cases there is an actual owner eventually and the primary task goes through the state transitions from
2223 *Created* to *Ready* to *Reserved* to *InProgress*, etc.

2224 In the second case where no actual owner is desired the human task (the primary task) directly transitions
2225 from state *Created* to *InProgress*. Subtasks are always instantiated automatically.

2226 **4.10.2 Releasing a Human Task**

2227 The current actual owner of a human task can *release* a task to again make it available for all potential
2228 owners. A task can be released from active states that have an actual owner (*Reserved*, *InProgress*),
2229 transitioning it into the *Ready* state. Business data associated with the task (intermediate result data, ad-
2230 hoc attachments and comments) is kept.

2231 A task that is currently *InProgress* can be stopped by the actual owner, transitioning it into state
2232 *Reserved*. Business data associated with the task as well as its actual owner is kept.

2233 **4.10.3 Delegating or Forwarding a Human Task**

2234 Task's potential owners, actual owner or business administrator can *delegate* a task to another user,
2235 making that user the actual owner of the task, and also adding her to the list of potential owners in case
2236 she is not, yet. A task can be delegated when it is in an active state (*Ready*, *Reserved*, *InProgress*), and
2237 transitions the task into the *Reserved* state. Business data associated with the task is kept.

2238 Similarly, task's potential owners, actual owner or business administrator can forward an active task to
2239 another person or a set of people, replacing himself by those people in the list of potential owners.
2240 Potential owners can only forward tasks that are in the *Ready* state. Forwarding is possible if the task has
2241 a set of individually assigned potential owners, not if its potential owners are assigned using one or many
2242 groups. If the task is in the *Reserved* or *InProgress* state then the task is implicitly released first, that is,
2243 the task is transitioned into the *Ready* state. Business data associated with the task is kept. The user
2244 performing the forward is removed from the set of potential owners of the task, and the forwarder is
2245 added to the set of potential owners.

2246 **4.10.4 Sub Task Event Propagation**

2247 Task state transitions may be caused by the invocation of API operations (see section 7 "Programming
2248 Interfaces") or by events (see section 8 "Interoperable Protocol for Advanced Interaction with Human
2249 Tasks").

2250 If a task has sub tasks then some state transitions are propagated to these sub tasks. Conversely, if a
 2251 task has a parent task then some state transitions are propagated to that parent task.

2252 The following table defines how task state transitions MUST be propagated to sub tasks and to parent
 2253 tasks.

Task Event	Effect on Sub Tasks (downward propagation)	Effect on Parent Task (upward propagation)
suspend operation invoked	suspend (ignored if not applicable, e.g., if the sub task is already suspended or in a final state) – a suspend event is propagated recursively if the sub task is not in a final state	none
suspend event received (from a parent task)		
resume operation invoked	resume (ignored if not applicable, e.g., if the sub task is not suspended or in a final state) – a resume event is propagated recursively if the sub task is not in a final state	none
resume event received (from a parent task)		
complete operation invoked	exit (ignored if the sub task is in a final state)	completion may be initiated (see section 4.7 “Completion Behavior”)
complete event received		
fail operation invoked	exit (ignored if the sub task is in a final state)	none (if “manual” activation pattern), otherwise fail
fail event received		
non-recoverable error event received		
exit event received	exit (ignored if the sub task is in a final state)	none
skip operation invoked (and the task is “skipable”)	skip	completion may be initiated (see section 4.7 “Completion Behavior”)

2254 All other task state transitions MUST NOT affect sub tasks or a parent task.

2255 **4.11 History of a Human Task**

2256 Task lifecycle state changes and data changes are maintained as a history of task events. Task events
 2257 contain the following data:

2258 **Task Event**

- 2259 • event id
- 2260 • event time
- 2261 • task id
- 2262 • user (principal) that caused the state change
- 2263 • event type (e.g. claim task).
- 2264 • event data (e.g. data used in setOutput) and fault name (event was setFault)
- 2265 • startOwner - the actual owner before the event.
- 2266 • endOwner - the actual owner after the event.
- 2267 • task status at the end of the event

2268 For example, if the User1 delegated a task to User2, then the user and startOwner would be User1,
 2269 endOwner would be User2. The event data would be the <htt:organizationalEntity/> element used in the
 2270 WSHT delegate operation.

2271 The system generated attribute 'event id' MUST be unique on a per task basis.

2272 4.11.1 Task Event Types and Data

2273 Some task events (e.g. setOutput) may have data associated with event and others may not (e.g. claim).
 2274 The following table lists the event types and the data.

Actions/Operations resulting in task events			
Event Type	Owner Change	State Change	Data Value
created	maybe	yes	
claim	yes	yes	
Start	maybe	yes	
stop		yes	
release	yes	yes	
suspend		yes	
suspendUntil		yes	<htt:pointOfTime>2020-12-12T12:12:12Z </htt:pointOfTime> or <htt:timePeriod>PT1H</htt:timePeriod>
resume		yes	
complete		yes	<htt:taskData> <ns:someData xmlns:ns="urn:foo"/> </htt:taskData>
remove			
fail		yes	<htt:fail> <htt:identifier>urn:b4p:1</htt:identifier> <htt:faultName>fault1</htt:faultName> <htt:faultData> <someFaultData xmlns="urn:foo"/> </htt:faultData> </htt:fail>
setPriority			<htt:priority>500000</htt:priority>
addAttachment			<htt:addAttachment> <htt:identifier>urn:b4p:1</htt:identifier> <htt:name>myAttachment</htt:name> <htt:accessType>MIME</htt:accessType> <htt:contentType>text/plain</htt:contentType> <htt:attachment/> </htt:addAttachment>

Actions/Operations resulting in task events			
Event Type	Owner Change	State Change	Data Value
deleteAttachment			<htt:identifier> urn:b4p:1</htt:identifier>
addComment			<htt:text>text for comment</htt:text>
updateComment			<htt:text>new text for comment</htt:text>
deleteComment			<htt:text>deleted comment text</htt:text>
skip		yes	
forward	maybe	maybe	<htt:organizationalEntity> <htt:user>user5</htt:user> <htt:user>user6</htt:user> </htt:organizationalEntity>
delegate	yes	maybe	<htt:organizationalEntity> <htt:user>user5</htt:user> </htt:organizationalEntity>
setOutput			<htt:setOutput> <htt:identifier>urn:b4p:1</htt:identifier> <htt:part>outputPart1</htt:part> <htt:taskData> <ns:someData xmlns:ns="urn:foo" /> </htt:taskData> </htt:setOutput>
deleteOutput			
setFault			<htt:setFault> <htt:identifier>urn:b4p:1</htt:identifier> <htt:faultName>fault1</htt:faultName> <htt:faultData><someFault xmlns="urn:fault" /></htt:faultData> </htt:setFault>
deleteFault			
activate	maybe	yes	
nominate	maybe	maybe	<htt:organizationalEntity> <htt:user>user1</htt:user> <htt:user>user2</htt:user> </htt:organizationalEntity>
setGenericHumanRole			<htt:setGenericHumanRole> <htt:identifier>urn:b4p:1</htt:identifier> <htt:genericHumanRole>businessAdministrators</htt:genericHumanRole> <htt:organizationalEntity> <htt:user>user7</htt:user> <htt:user>user8</htt:user> </htt:organizationalEntity> </htt:setGenericHumanRole>

Actions/Operations resulting in task events			
Event Type	Owner Change	State Change	Data Value
expire		yes	
escalated			
cancel			

2275 4.11.2 Retrieving the History

2276 There is a `getTaskHistory` operation that allows a client to query the system and retrieve a list of task
2277 events that represent the history of the task. This operation can:

- 2278 • Return a list of task events with optional data
- 2279 • Return a list of task events without optional event data
- 2280 • Return a subset of the events based on a range (for paging)
- 2281 • Return a filtered list of events.

2282 The option to whether or not to include event data is useful since in some cases the event data content
2283 (e.g. `setOutput`) may be large. In a typical case, an API client should be able to query the system to get a
2284 "light weight" response of events (e.g. with out event data) and then when necessary, make an additional
2285 API call to get a specific event details with data. The latter can be accomplished by specifying the event id
2286 when invoking the `getTaskHistory` operation.

2287 The XML Schema definition of the filter is the following:

```

2288 <xsd:complexType name="tTaskHistoryFilter">
2289   <xsd:choice>
2290     <xsd:element name="eventId" type="xsd:integer" />
2291     <!-- Filter to allow narrow down query by status, principal,
2292          event Type. -->
2293     <xsd:sequence>
2294       <xsd:element name="status" type="tStatus" minOccurs="0"
2295                   maxOccurs="unbounded" />
2296       <xsd:element name="eventType" type="tTaskEventType" minOccurs="0"
2297                   maxOccurs="unbounded" />
2298       <xsd:element name="principal" type="xsd:string" minOccurs="0" />
2299       <xsd:element name="afterEventTime" type="xsd:dateTime"
2300                   minOccurs="0" />
2301       <xsd:element name="beforeEventTime" type="xsd:dateTime"
2302                   minOccurs="0" />
2303     </xsd:sequence>
2304   </xsd:choice>
2305 </xsd:complexType>
2306
2307 <xsd:simpleType name="tTaskEventType">
2308   <xsd:restriction base="xsd:string">
2309     <xsd:enumeration value="create" />
2310     <xsd:enumeration value="claim" />
2311     <xsd:enumeration value="start" />
2312     <xsd:enumeration value="stop" />
2313     <xsd:enumeration value="release" />
2314     <xsd:enumeration value="suspend" />

```

```

2315     <xsd:enumeration value="suspendUntil" />
2316     <xsd:enumeration value="resume" />
2317     <xsd:enumeration value="complete" />
2318     <xsd:enumeration value="remove" />
2319     <xsd:enumeration value="fail" />
2320     <xsd:enumeration value="setPriority" />
2321     <xsd:enumeration value="addAttachment" />
2322     <xsd:enumeration value="deleteAttachment" />
2323     <xsd:enumeration value="addComment" />
2324     <xsd:enumeration value="updateComment" />
2325     <xsd:enumeration value="deleteComment" />
2326     <xsd:enumeration value="skip" />
2327     <xsd:enumeration value="forward" />
2328     <xsd:enumeration value="delegate" />
2329     <xsd:enumeration value="setOutput" />
2330     <xsd:enumeration value="deleteOutput" />
2331     <xsd:enumeration value="setFault" />
2332     <xsd:enumeration value="deleteFault" />
2333     <xsd:enumeration value="activate" />
2334     <xsd:enumeration value="nominate" />
2335     <xsd:enumeration value="setGenericHumanRole" />
2336     <xsd:enumeration value="expire" />
2337     <xsd:enumeration value="escalated" />
2338   </xsd:restriction>
2339 </xsd:simpleType>

```

2340 The XML Schema definition of events returned for the history is the following:

```

2341 <xsd:element name="taskEvent">
2342   <xsd:complexType>
2343     <xsd:annotation>
2344       <xsd:documentation>
2345         A detailed event that represents a change in the task's state.
2346       </xsd:documentation>
2347     </xsd:annotation>
2348     <xsd:sequence>
2349       <!-- event id - unique per task -->
2350       <xsd:element name="id" type="xsd:integer" />
2351       <!-- event date time -->
2352       <xsd:element name="eventTime" type="xsd:dateTime" />
2353       <!-- task ID -->
2354       <xsd:element name="identifier" type="xsd:anyURI" />
2355       <xsd:element name="principal" type="xsd:string" minOccurs="0"
2356         nillable="true" />
2357       <!-- Event type. Note - using a restricted type limits
2358         extensibility to add custom event types. -->
2359       <xsd:element name="eventType" type="tTaskEventType" />
2360       <!-- actual owner of the task before the event -->
2361       <xsd:element name="startOwner" type="xsd:string" minOccurs="0"
2362         nillable="true" />
2363       <!-- actual owner of the task after the event -->
2364       <xsd:element name="endOwner" type="xsd:string" minOccurs="0"
2365         nillable="true" />
2366       <!-- WSHT task status -->
2367       <xsd:element name="status" type="tStatus" />
2368       <!-- boolean to indicate this event has optional data -->
2369       <xsd:element name="hasData" type="xsd:boolean" minOccurs="0" />
2370       <xsd:element name="eventData" type="xsd:anyType" minOccurs="0"

```

```
2371         nillable="true" />
2372     <xsd:element name="faultName" type="xsd:string" minOccurs="0"
2373         nillable="true" />
2374     <!-- extensibility -->
2375     <xsd:any namespace="##other" processContents="lax" minOccurs="0"
2376         maxOccurs="unbounded" />
2377 </xsd:sequence>
2378 </xsd:complexType>
2379 </xsd:element>
2380
```

2381 5 Lean Tasks

2382 The <leanTask> element is used to specify human tasks. This section introduces the syntax for the
2383 element, and individual properties are explained in subsequent sections.

2384 5.1 Overall Syntax

2385 The element <leanTask> derives from the type htd:tTask, with the following augmentations:

```
2386 <htd:leanTask>  
2387   <htd:interface>...</htd:interface>  
2388   <htd:messageSchema>...</htd:messageSchema>  
2389   ... All elements from htd:task except <interface> and <composition> ...  
2390   <htd:composition>...</htd:composition>  
2391 </htd:leanTask>
```

2392 5.2 Properties

2393 The following attributes and elements are defined for lean tasks and are different from the definition of
2394 htd:task:

- 2395 • interface – Lean tasks are created through the CreateLeanTask operation (section 7.3.4), and
2396 their input message is derived from the messageSchema element. Therefore, an interface
2397 element might contradict that information, and to prevent that, interface is banned.
- 2398 • messageSchema – Identifies the schema of the inputMessage and outputMessage for the lean
2399 task, and if the renderings element is not defined, the WS-HumanTask Processor can use this to
2400 generate a rendering or pass this data directly to a WS-HumanTask Client such that the
2401 rendering is generated from the messageSchema.
- 2402 • composition – Lean tasks cannot have explicitly declared subtasks as defined for composite
2403 tasks (section 4.6), consequently, this element is banned.

2404 5.3 Message Schema

2405 This element references the schema of the data that is used for both the input and output messages of
2406 the lean task.

```
2407 <messageSchema>  
2408   <messageField name="xsd:NCName" type="xsd:QName">*  
2409     <messageDisplay xml:lang="xsd:language"?>+  
2410       Language specific display  
2411     </messageDisplay>  
2412     <messageChoice name="xsd:NCName">*  
2413       <messageDisplay xml:lang="xsd:language"?>+  
2414         Language specific display  
2415       </messageDisplay>  
2416     </messageChoice>  
2417   </messageField>  
2418 </messageSchema>
```

2419 The <messageSchema> element specifies the data that a Lean Task accepts. As it is currently defined, a
2420 WS-HumanTask Processor could render the following form elements in a way that only requires vendor-
2421 specific knowledge between the WS-HumanTask Processor and the WS-HumanTask Client and no
2422 vender-specific knowledge between the WS-HumanTask Processor and the WS-HumanTask Parent:

- 2423 • String
- 2424 • Integer

- 2425 • Float
- 2426 • Date Time
- 2427 • Bool
- 2428 • Enumeration (Choice)

2429 Each of these is accomplished by using an instance of a `<messageField>`. For string, integer, float,
2430 datetime, and boolean fields, this is accomplished by using the type attribute of the `<messageField>`.
2431 The supported set of values are: `xsd:string`, `xsd:integer`, `xsd:float`, `xsd:datetime`, and
2432 `xsd:boolean`, all respectively matching the list above. If a simple rendering language like HTML were
2433 used, this could be accomplished by using a textbox control that simply had special rules about the format
2434 of its input.

2435 The enumeration field uses a combination of one `<messageField>` element and possibly many child
2436 `<messageChoice>` elements. Each child `<messageChoice>` represents one possible option that could
2437 be selected from the enumeration. If a simple rendering language like HTML were used, this could be
2438 shown using radio buttons, a dropdown list, or a listbox that only supports single selection.

2439 For all `<messageField>` and `<messageChoice>` elements, it is possible to specify a per-language
2440 `<messageDisplay>` element. It uses `xml:lang`, a standard XML attribute, to define the language of the
2441 enclosed information. This attribute uses tags according to RFC 1766 (see [RFC1766]). There could be
2442 zero or more `<messageDisplay>` elements. A `<messageField>` or `<messageChoice>` MUST NOT
2443 specify multiple `<messageDisplay>` elements having the same value for the attribute `xml:lang`.

2444 The combination of `<messageSchema>` and `<possibleOutcomes>` can be used to generate a form of
2445 sufficient functionality for many simple tasks, precluding the need for a renderings element.

2446 **Example:**

```
2447 <messageSchema>
2448   <messageField name="amount" type="xsd:float">
2449     <messageDisplay xml:lang="en-us">Amount</messageDisplay>
2450     <messageDisplay xml:lang="fr-fr">Quantité</messageDisplay>
2451   </messageField>
2452   <messageField name="currencyUnit" type="xsd:string">
2453     <messageDisplay xml:lang="en-us">Currency</messageDisplay>
2454     <messageDisplay xml:lang="fr-fr">Devise</messageDisplay>
2455     <messageChoice name="USD">
2456       <messageDisplay xml:lang="en-us">US Dollars</messageDisplay>
2457       <messageDisplay xml:lang="fr-fr">US Dollars</messageDisplay>
2458     </messageChoice>
2459     <messageChoice name="EURO">
2460       <messageDisplay xml:lang="en-us">Euro Dollars</messageDisplay>
2461       <messageDisplay xml:lang="fr-fr">Euros</messageDisplay>
2462     </messageChoice>
2463   </messageField>
2464 </messageSchema>
```

2465

2466

2467 **5.4 Example: ToDoTask**

2468 The following XML could be used for a simple 'ToDoTask':

```
2469 <htd:task name="ToDoTask">
2470   <htd:messageSchema />
2471   <htd:possibleOutcomes>
2472     <htd:possibleOutcome name="Completed" />
2473     ... language specific translations ...
2474   </htd:possibleOutcomes>
2475   <htd:delegation potentialDelegates="anybody" />
2476   <htd:presentationElements>
2477     <htd:name>ToDo Task</htd:name>
2478     ... language specific translations ...
2479     <htd:subject>Please complete the described work</htd:subject>
2480     ... language specific translations ...
2481     <htd:description contentType="mimeTypeString" />
2482     ... language specific translations ...
2483   </htd:presentationElements>
2484 </htd:task>
```


2485

6 Notifications

2486 Notifications are used to notify a person or a group of people of a noteworthy business event, such as
2487 that a particular order has been approved, or a particular product is about to be shipped. They are also
2488 used in escalation actions to notify a user that a task is overdue or a task has not been started yet. The
2489 person or people to whom the notification will be assigned to could be provided, for example, as result of
2490 a people query to organizational model.

2491 Notifications are simple human interactions that do not block the progress of the caller, that is, the caller
2492 does not wait for the notification to be completed. Moreover, the caller cannot influence the execution of
2493 notifications, e.g. notifications are not terminated if the caller terminates. The caller, i.e. an application, a
2494 business process or an escalation action, initiates a notification passing the required notification data. The
2495 notification appears on the task list of all notification recipients. After a notification recipient removes it,
2496 the notification disappears from the recipient's task list.

2497 A notification MAY have multiple recipients and optionally one or many business administrators. The
2498 generic human roles task initiator, task stakeholders, potential owners, actual owner and excluded
2499 owners play no role.

2500 Presentation elements and task rendering, as described in sections 4.3 and 4.4 respectively, are used for
2501 notifications also. In most cases the subject line and description are sufficient information for the
2502 recipients, especially if the notifications are received in an e-mail client or mobile device. But in some
2503 cases the notifications can be received in a proprietary client so the notification can support a proprietary
2504 rendering format to enable this to be utilized to the full, such as for rendering data associated with the
2505 caller invoking the notification. For example, the description could include a link to the process audit trail
2506 or a button to navigate to business transactions involved in the underlying process.

2507 Notifications do not have ad-hoc attachments, comments or deadlines.

2508 6.1 Overall Syntax

2509 Definition of notifications

```
2510 <htd:notification name="NCName" >  
2511  
2512   <htd:interface portType="QName" operation="NCName" />  
2513  
2514   <htd:priority expressionLanguage="anyURI"??>  
2515     integer-expression  
2516   </htd:priority>  
2517  
2518   <htd:peopleAssignments>  
2519  
2520     <htd:recipients>  
2521       ...  
2522     </htd:recipients>  
2523  
2524     <htd:businessAdministrators?>  
2525       ...  
2526     </htd:businessAdministrators>  
2527  
2528   </htd:peopleAssignments>  
2529  
2530   <htd:presentationElements>  
2531     ...  
2532   </htd:presentationElements>  
2533  
2534   <htd:renderings?>
```

```
2535     ...
2536     </htd:renderings>
2537
2538 </htd:notification>
```

2539 6.2 Properties

2540 The following attributes and elements are defined for notifications:

- 2541 • `name`: This attribute is used to specify the name of the notification. The name combined with the
2542 target namespace **MUST** uniquely identify a notification in the notification definition. The attribute
2543 is mandatory. It is not used for notification rendering.
- 2544 • `interface`: This element is used to specify the operation used to invoke the notification. The
2545 operation is specified using WSDL, that is a WSDL port type and WSDL operation are defined.
2546 The element and its `portType` and `operation` attributes are mandatory. In the `operation`
2547 attribute, a WS-HumanTask Definition **MUST** reference a one-way WSDL operation.
- 2548 • `priority`: This element is used to specify the priority of the notification. It is an optional
2549 element which value is an integer expression. If present then the WS-HumanTask Definition
2550 **MUST** specify a value between 0 and 10, where 0 is the highest priority and 10 is the lowest. If
2551 not present, the priority of the notification is considered as 5. The result of the expression
2552 evaluation is of type `htt:tPriority`. The `expressionLanguage` attribute specifies the
2553 language used in the expression. The attribute is optional. If not specified, the default language
2554 as inherited from the closest enclosing element that specifies the attribute is used.
- 2555 • `peopleAssignments`: This element is used to specify people assigned to the notification. The
2556 element is mandatory. A WS-HumanTask Definition **MUST** include a people assignment for
2557 recipients and **MAY** include a people assignment for business administrators.
- 2558 • `presentationElements`: The element is used to specify different information used to display
2559 the notification, such as name, subject and description, in a task list. The element is mandatory.
2560 See section 4.3 for more information on presentation elements.
- 2561 • `rendering`: The element is used to specify rendering method. It is optional. If not present,
2562 notification rendering is implementation dependent. See section 4.4 for more information on
2563 rendering.

2564 6.3 Notification Behavior and State Transitions

2565 Same as human tasks, notifications are in pseudo-state *Inactive* before they are activated. Once they are
2566 activated they move to the *Ready* state. This state is observable, that is, when querying for notifications
2567 then all notifications in state *Ready* are returned. When a notification is removed then it moves into the
2568 final pseudo-state *Removed*.

2569

7 Programming Interfaces

2570

7.1 Operations for Client Applications

2571

A number of applications are involved in the life cycle of a task. These comprise:

2572

- The task list client, i.e. a client capable of displaying information about the task under consideration

2573

2574

- The requesting application, i.e. any partner that has initiated the task

2575

- The supporting application, i.e. an application launched by the task list client to support processing of the task.

2576

2577

The task infrastructure provides access to a given task. It is important to understand that what is meant by *task list client* is the software that presents a UI to one authenticated user, irrespective of whether this UI is rendered by software running on server hardware (such as in a portals environment) or client software (such as a client program running on a users workstation or PC).

2578

2579

2580

2581

A given task exposes a set of operations to this end. A WS-HumanTask Processor MUST provide the operations listed below and an application (such as a task list client) can use these operations to manipulate the task. All operations MUST be executed in a synchronous fashion and MUST return a fault if certain preconditions do not hold. For operations that are not expected to return a response they MAY return a void message. The above applies to notifications also.

2582

2583

2584

2585

2586

An operation takes a well-defined set of parameters as its input. Passing an illegal parameter or an illegal number of parameters MUST result in the `hta:illegalArgumentFault` being returned. Invoking an operation that is not allowed in the current state of the task MUST result in an `hta:illegalStateFault`.

2587

2588

2589

2590

By default, the identity of the person on behalf of which the operation is invoked is passed to the task.

2591

When the person is not authorized to perform the operation the `hta:illegalAccessFault` and `hta:recipientNotAllowed` MUST be returned in the case of tasks and notifications respectively.

2592

2593

Invoking an operation that does not apply to the task type (e.g., invoking claim on a notification) MUST result in an `hta:illegalOperationFault`.

2594

2595

The language of the person on behalf of which the operation is invoked is assumed to be available to operations requiring that information, e.g., when accessing presentation elements.

2596

2597

For an overview of which operations are allowed in what state, refer to section 4.10 "Human Task Behavior and State Transitions". For a formal definition of the allowed operations, see Appendix D "WS-HumanTask Client API Port Type".

2598

2599

2600

For information which generic human roles are authorized to perform which operations, refer to section 7.1.4 "Operation Authorizations".

2601

2602

This specification does not stipulate the authentication, language passing, addressing, and binding scheme employed when calling an operation. This can be achieved using different mechanisms (e.g. WS-Security, WS-Addressing).

2603

2604

2605

7.1.1 Participant Operations

2606

Operations are executed by end users, i.e. actual or potential owners. The identity of the user is implicitly passed when invoking any of the operations listed in the table below.

2607

2608

If the task is in a predefined state listed as valid pre-state before the operation is invoked then, upon successful completion, the task MUST be in the post state defined for the operation. If the task is in a predefined state that is not listed as valid pre-state before the operation is invoked then the operation MUST be rejected and MUST NOT cause a task state transition.

2609

2610

2611

2612

All of the operations below apply to tasks and sub tasks only unless specifically noted below.

2613 The column “**Supports Batch Processing**” below indicates if an operation can be used to process
 2614 multiple human tasks at the same time. One or more operations on individual tasks may fail without
 2615 causing the overall batch operation to fail.
 2616

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
claim	Claim responsibility for a task, i.e. set the task to status <i>Reserved</i>	In <ul style="list-style-type: none"> task identifier Out <ul style="list-style-type: none"> void 	Yes	Ready	Reserved
start	Start the execution of the task, i.e. set the task to status <i>InProgress</i> .	In <ul style="list-style-type: none"> task identifier Out <ul style="list-style-type: none"> void 	Yes	Ready Reserved	InProgress
stop	Cancel/stop the processing of the task. The task returns to the <i>Reserved</i> state.	In <ul style="list-style-type: none"> task identifier Out <ul style="list-style-type: none"> void 	Yes	InProgress	Reserved
release	Release the task, i.e. set the task back to status <i>Ready</i> .	In <ul style="list-style-type: none"> task identifier Out <ul style="list-style-type: none"> void 	Yes	InProgress Reserved	Ready
suspend	Suspend the task.	In <ul style="list-style-type: none"> task identifier Out <ul style="list-style-type: none"> void 	Yes	Ready Reserved InProgress	Suspended/ Ready (from Ready) Suspended/ Reserved (from Reserved) Suspended/ InProgress (from InProgress)
suspendUntil	Suspend the task for a given period of time or until a fixed point in time. The WS-HumanTask Client MUST specify either a period of time or a fixed point in time.	In <ul style="list-style-type: none"> task identifier time period point of time Out <ul style="list-style-type: none"> void 	Yes	Ready Reserved InProgress	Suspended/ Ready (from Ready) Suspended/ Reserved (from Reserved) Suspended/ InProgress (from InProgress)

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
resume	Resume a suspended task.	In <ul style="list-style-type: none"> task identifier Out <ul style="list-style-type: none"> void 	Yes	Suspended/Ready Suspended/Reserved Suspended/InProgress	Ready (from Suspended/Ready) Reserved (from Suspended/Reserved) InProgress (from Suspended/InProgress)
complete	Execution of the task finished successfully. If no output data is set the operation MUST return <code>hta:illegalArgumentFault</code> .	In <ul style="list-style-type: none"> task identifier output data of task Out <ul style="list-style-type: none"> void 	Yes	InProgress	Completed
remove	Applies to notifications only. Used by notification recipients to remove the notification permanently from their task list client. It will not be returned on any subsequent retrieval operation invoked by the same user.	In <ul style="list-style-type: none"> task identifier Out <ul style="list-style-type: none"> void 	Yes	Ready (Notification state)	Removed (Notification state)
fail	Execution of the task fails and a fault is returned. The fault <code>hta:illegalOperationFault</code> MUST be returned if the task interface defines no faults. If fault name or fault data is not set the operation MUST return <code>hta:illegalArgumentFault</code> .	In <ul style="list-style-type: none"> task identifier fault – contains the fault name and fault data Out <ul style="list-style-type: none"> void 	Yes	InProgress	Failed

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
setPriority	Change the priority of the task. The WS-HumanTask Client MUST specify the integer value of the new priority.	In <ul style="list-style-type: none"> task identifier priority (<code>htt:tPriority</code>) Out <ul style="list-style-type: none"> void 	Yes	(any state)	(no state transition)
addAttachment	Add attachment to a task. Returns an identifier for the attachment.	In <ul style="list-style-type: none"> task identifier attachment name access type content type attachment Out <ul style="list-style-type: none"> attachment identifier 	No	(any state)	(no state transition)
getAttachmentInfos	Get attachment information for all attachments associated with the task.	In <ul style="list-style-type: none"> task identifier Out <ul style="list-style-type: none"> list of attachment data (list of <code>htt:attachmentInfo</code>) 	No	(any state)	(no state transition)
getAttachment	Get the task attachment with the given identifier.	In <ul style="list-style-type: none"> task identifier attachment identifier Out <ul style="list-style-type: none"> <code>htt:attachment</code> 	No	(any state)	(no state transition)
deleteAttachment	Delete the attachment with the specified identifier from the task. Attachments provided by the enclosing context MUST NOT be affected by this operation.	In <ul style="list-style-type: none"> task identifier attachment identifier Out <ul style="list-style-type: none"> void 	No	(any state)	(no state transition)

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
addComment	Add a comment to a task. Returns an identifier that can be used to later update or delete the comment.	In <ul style="list-style-type: none"> task identifier plain text Out <ul style="list-style-type: none"> comment identifier 	No	(any state)	(no state transition)
updateComment	Updates the identified comment with the supplied new text.	In <ul style="list-style-type: none"> task identifier comment identifier plain text Out <ul style="list-style-type: none"> void 	No	(any state)	(no state transition)
deleteComment	Deletes the identified comment.	In <ul style="list-style-type: none"> task identifier comment identifier Out <ul style="list-style-type: none"> void 	No	(any state)	(no state transition)
getComments	Get all comments of a task	In <ul style="list-style-type: none"> task identifier Out <ul style="list-style-type: none"> list of comments (list of <code>htt:comment</code>) 	No	(any state)	(no state transition)
skip	Skip the task. If the task is not skipable then the fault <code>hta:illegalOperationFault</code> MUST be returned.	In <ul style="list-style-type: none"> task identifier Out <ul style="list-style-type: none"> void 	Yes	Created Ready Reserved InProgress	Obsolete

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
forward	Forward the task to another organization entity. The WS-HumanTask Client MUST specify the receiving organizational entity. Potential owners MAY forward a task while the task is in the <i>Ready</i> state. For details on forwarding human tasks refer to section 4.10.3.	In <ul style="list-style-type: none"> task identifier organizational entity (<code>http://tOrganizationalEntity</code>) Out <ul style="list-style-type: none"> void 	Yes	Ready Reserved InProgress	Ready
delegate	Assign the task to one user and set the task to state <i>Reserved</i> . If the recipient was not a potential owner then this person MUST be added to the set of potential owners. For details on delegating human tasks refer to section 4.10.3.	In <ul style="list-style-type: none"> task identifier organizational entity (<code>http://tOrganizationalEntity</code>) Out <ul style="list-style-type: none"> void 	Yes	Ready Reserved InProgress	Reserved
getRendering	Applies to both tasks and notifications. Returns the rendering specified by the type parameter.	In <ul style="list-style-type: none"> task identifier rendering type Out <ul style="list-style-type: none"> any type 	No	(any state)	(no state transition)
getRenderingTypes	Applies to both tasks and notifications. Returns the rendering types available for the task or notification.	In <ul style="list-style-type: none"> task identifier Out <ul style="list-style-type: none"> list of QNames 	No	(any state)	(no state transition)
getTaskDetails	Applies to both tasks and notifications. Returns a data object of type <code>http://tTaskDetails</code>	In <ul style="list-style-type: none"> task identifier Out <ul style="list-style-type: none"> task (<code>http://tTaskDetails</code>) 	No	(any state)	(no state transition)

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
getTaskDescription	Applies to both tasks and notifications. Returns the presentation description in the specified mime type.	In <ul style="list-style-type: none"> task identifier content type – optional, default is text/plain Out <ul style="list-style-type: none"> string 	No	(any state)	(no state transition)
getTaskOperations	Applies to tasks. Returns list of operations that are available to the authorized user given the user's role and the state of the task.	In <ul style="list-style-type: none"> task identifier Out <ul style="list-style-type: none"> List of available operation. 	No	(any state)	(no state transition)
setOutput	Set the data for the part of the task's output message.	In <ul style="list-style-type: none"> task identifier part name (optional for single part messages) output data of task Out <ul style="list-style-type: none"> void 	No	InProgress	(no state transition)
deleteOutput	Deletes the output data of the task.	In <ul style="list-style-type: none"> task identifier Out <ul style="list-style-type: none"> void 	No	InProgress	(no state transition)
setFault	Set the fault data of the task. The fault <code>hta:illegalOperationFault</code> MUST be returned if the task interface defines no faults.	In <ul style="list-style-type: none"> task identifier fault – contains the fault name and fault data Out <ul style="list-style-type: none"> void 	No	InProgress	(no state transition)
deleteFault	Deletes the fault name and fault data of the task.	In <ul style="list-style-type: none"> task identifier Out <ul style="list-style-type: none"> void 	No	InProgress	(no state transition)

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
getInput	Get the data for the part of the task's input message.	In <ul style="list-style-type: none"> task identifier part name (optional for single part messages) Out <ul style="list-style-type: none"> any type 	No	(any state)	(no state transition)
getOutput	Get the data for the part of the task's output message.	In <ul style="list-style-type: none"> task identifier part name (optional for single part messages) Out <ul style="list-style-type: none"> any type 	No	(any state)	(no state transition)
getFault	Get the fault data of the task.	In <ul style="list-style-type: none"> task identifier Out <ul style="list-style-type: none"> fault – contains the fault name and fault data 	No	(any state)	(no state transition)
getOutcome	Get the outcome of the task	In <ul style="list-style-type: none"> task identifier Out <ul style="list-style-type: none"> string 	No	(any state)	(no state transition)

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
getTaskHistory	<p>Get a list of events representing the history of the task.</p> <p><i>Filter</i> allows narrowing the results by status, principal, event Type.</p> <p><i>startIndex</i> and <i>maxTasks</i> are integers that allow paging of the results.</p> <p><i>includeData</i> is a Boolean. Data is included with the returned events only if this is true.</p>	<p>In</p> <ul style="list-style-type: none"> • task identifier • filter (htt:tTaskHistoryFilter) • startIndex • maxTasks • includeData <p>Out</p> <ul style="list-style-type: none"> • list of htt:taskEvent 	No	(any state)	(no state transition)

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
getTaskInstanceData	<p>Get any or all details of a task, except the contents of the attachments. This duplicates functionality provided by the get() operations above, but provides all the data in a single round trip.</p> <p><i>Properties</i> is an optional space separated list of properties of the task that should be provided. Properties are named by the local part of the QName of the element returned for task details.</p> <p>If it is not specified, then all properties are returned.</p> <p>If it is specified, then only the properties specified are returned. In the case that multiple elements have the same local part (which can happen for extensions from two different namespaces) then all of the matching properties are returned.</p> <p>Some properties of a task may have multiple values (i.e., taskDescription, input and output). When such a property is requested, all valid values for the property are returned. There is an exception for the "renderings" property, which is controlled by the "renderingPreference" parameter.</p> <p><i>renderingPreference</i> is an optional list of rendering types, in order of preference. If</p>	<p>In</p> <ul style="list-style-type: none"> • task identifier • properties • rendering preferences <p>Out</p> <ul style="list-style-type: none"> • task (http://TaskInstanceData) 	No	(any state)	(no state transition)

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
getSubtasks	Returns all sub tasks of a task (created instances + not yet created sub task definitions)	In <ul style="list-style-type: none"> task identifier Out <ul style="list-style-type: none"> list of tasks (list of htt:tTask) 	No	(any state)	(no state transition)
getSubtaskIdentifiers	Returns the identifiers of all already created sub tasks of a task	In <ul style="list-style-type: none"> task identifier Out <ul style="list-style-type: none"> list of task identifiers 	No	(any state)	(no state transition)
hasSubtasks	Returns true if a task has at least one (already created or not yet created, but specified) sub task	In <ul style="list-style-type: none"> task identifier Out <ul style="list-style-type: none"> boolean 	No	(any state)	(no state transition)
getParentTask	Returns the superior composite task of a sub task	In <ul style="list-style-type: none"> task identifier Out <ul style="list-style-type: none"> htt:tTask 	No	(any state)	(no state transition)
getParentTaskIdentifier	Returns the task identifier of the superior composite task of a sub task	In <ul style="list-style-type: none"> task identifier Out <ul style="list-style-type: none"> task identifier 	No	(any state)	(no state transition)
isSubtask	Returns true if a task is a sub task of a superior composite task	In <ul style="list-style-type: none"> task identifier Out <ul style="list-style-type: none"> boolean 	No	(any state)	(no state transition)

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
instantiateSubTask	<p>Creates an instantiateable subtask for the task from the definition of the task.</p> <p>The fault hta:illegalArgumentFault MUST be returned if the task does not have an instantiateable subtask of the given name.</p> <p>Returns the identifier for the created subtask.</p>	<p>In</p> <ul style="list-style-type: none"> task identifier subtask name <p>Out</p> <ul style="list-style-type: none"> task identifier 	No	Reserved In Progress	(no state transition)
setTaskStartDeadlineExpression	Sets a deadline expression for the named start deadline of the task	<p>In</p> <ul style="list-style-type: none"> task identifier deadline name deadline expression <p>Out</p> <ul style="list-style-type: none"> void 	Yes	Created Ready Reserved In Progress	(no state transition)
setTaskStartDurationExpression	Sets a duration expression for the named start deadline of the task	<p>In</p> <ul style="list-style-type: none"> task identifier deadline name duration expression <p>Out</p> <p>void</p>	Yes	Created Ready Reserved In Progress	(no state transition)
setTaskCompletionDeadlineExpression	Sets a deadline expression for the named completion deadline of the task	<p>In</p> <ul style="list-style-type: none"> task identifier deadline name deadline expression <p>Out</p> <p>void</p>	Yes	Created Ready Reserved In Progress	(no state transition)

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
setTaskCompletionDurationExpression	Sets a duration expression for the named completion deadline of the task	In <ul style="list-style-type: none"> task identifier deadline name duration expression Out void	Yes	Created Ready Reserved In Progress	(no state transition)

2617

2618 7.1.2 Simple Query Operations

2619 Simple query operations allow retrieving task data. These operations MUST be supported by a WS-
 2620 HumanTask Processor. The identity of the user is implicitly passed when invoking any of the following
 2621 operations.

2622 The following operations will return both matching tasks and sub tasks.

2623

Operation Name	Description	Parameters	Authorization
getMyTaskAbstracts	<p>Retrieve the task abstracts. This operation is used to obtain the data required to display a task list.</p> <p>If no task type has been specified then the default value "ALL" MUST be used.</p> <p>If no generic human role has been specified then the default value "actualOwner" MUST be used.</p> <p>If no work queue has been specified then only personal tasks MUST be returned. If the work queue is specified then only tasks of that work queue MUST be returned.</p> <p>If no status list has been specified then tasks in all valid states are returned.</p> <p>The where clause is</p>	In <ul style="list-style-type: none"> task type ("ALL" "TASKS" "NOTIFICATIONS") generic human role work queue status list where clause order-by clause created-on clause maxTasks taskIndexOffset Out <ul style="list-style-type: none"> list of tasks (list of <code>htt:tTaskAbstract</code>) 	Any

Operation Name	Description	Parameters	Authorization
	<p>optional. If specified, it MUST reference exactly one column using the following operators: <i>equals</i> (“=”), <i>not equals</i> (“<>”), <i>less than</i> (“<”), <i>greater than</i> (“>”), <i>less than or equals</i> (“<=”), and <i>greater than or equals</i> (“>=”), e.g., “Task.Priority = 1”).</p> <p>The created-on clause is optional. The <i>where</i> clause is logically ANDed with the created-on clause, which MUST reference the column Task.CreatedTime with operators as described above. The combination of the two clauses enables simple but restricted paging in a task list client.</p> <p>If maxTasks is specified, then the number of task abstracts returned for this query MUST NOT exceed this limit. The taskIndexOffset can be used to perform multiple identical queries and iterate over result sets where the maxTasks size exceeds the query limit. If maxTasks has not been specified then all tasks fulfilling the query are returned.</p>		
getMyTaskDetails	<p>Retrieve the task details. This operation is used to obtain the data required to display a task list, as well as the details for the individual tasks.</p> <p>If no task type has been specified then</p>	<p>In</p> <ul style="list-style-type: none"> • task type (“ALL” “TASKS” “NOTIFICATIONS”) • generic human role • work queue • status list 	Any

Operation Name	Description	Parameters	Authorization
	<p>the default value "ALL" MUST be used.</p> <p>If no generic human role has been specified then the default value "actualOwner" MUST be used.</p> <p>If no work queue has been specified then only personal tasks MUST be returned. If the work queue is specified then only tasks of that work queue MUST be returned.</p> <p>If no status list has been specified then tasks in all valid states are returned.</p> <p>The where clause is optional. If specified, it MUST reference exactly one column using the following operators: <i>equals</i> ("="), <i>not equals</i> ("<>"), <i>less than</i> ("<"), <i>greater than</i> (">"), <i>less than or equals</i> ("<="), and <i>greater than or equals</i> (">="), e.g., "Task.Priority = 1".</p> <p>The created-on clause is optional. The <i>where</i> clause is logically ANDed with the created-on clause, which MUST reference the column Task.CreatedTime with operators as described above. The combination of the two clauses enables simple but restricted paging in the task list client.</p> <p>If maxTasks is specified, then the number of task details returned for this query</p>	<ul style="list-style-type: none"> • where clause • created-on clause • maxTasks <p>Out</p> <ul style="list-style-type: none"> • list of tasks (list of <code>htt:tTaskDetails</code>) 	

Operation Name	Description	Parameters	Authorization
	MUST NOT exceed this limit. If maxTasks has not been specified then all tasks fulfilling the query are returned.		

2624

2625 The return types `tTaskAbstract` and `tTaskDetails` are defined in section 3.8.4 “Data Types for Task
2626 Instance Data”.

2627 **Simple Task View**

2628 The table below lists the task attributes available to the simple query operations. This view is used when
2629 defining the where clause of any of the above query operations.

2630

Column Name	Type
ID	xsd:string
TaskType	Enumeration
Name	xsd:QName
Status	Enumeration (for values see 4.10 “Human Task Behavior and State Transitions”)
Priority	htt:tPriority
CreatedTime	xsd:dateTime
ActivationTime	xsd:dateTime
ExpirationTime	xsd:dateTime
HasPotentialOwners	xsd:boolean
StartByTimeExists	xsd:boolean
CompleteByTimeExists	xsd:boolean
RenderingMethodExists	xsd:boolean
Escalated	xsd:boolean
ParentTaskId	xsd:string
HasSubTasks	xsd:boolean

Column Name	Type
SearchBy	xsd:string
Outcome	xsd:string

2631

2632 7.1.3 Advanced Query Operation

2633 The advanced query operation is used by the task list client to perform queries not covered by the simple
 2634 query operations defined in 7.1.2. A WS-HumanTask Processor MAY support this operation. An
 2635 implementation MAY restrict the results according to authorization of the invoking user.

2636

2637 The following operations will return both matching tasks and sub tasks.

2638

Operation Name	Description	Parameters
query	Retrieve task data. All clauses assume a (pseudo-) SQL syntax. If maxTasks is specified, then the number of task returned by the query MUST NOT exceed this limit. The taskIndexOffset can be used to perform multiple identical queries and iterate over result sets where the maxTasks size exceeds the query limit.	In <ul style="list-style-type: none"> • select clause • where clause • order-by clause • maxTasks • taskIndexOffset Out <ul style="list-style-type: none"> • task query result set (htt:tTaskQueryResultSet)

2639

2640 ResultSet Data Type

2641 This is the result set element that is returned by the query operation.

```

2642 <xsd:element name="taskQueryResultSet" type="tTaskQueryResultSet" />
2643 <xsd:complexType name="tTaskQueryResultSet">
2644   <xsd:sequence>
2645     <xsd:element name="row" type="tTaskQueryResultRow"
2646       minOccurs="0" maxOccurs="unbounded" />
2647   </xsd:sequence>
2648 </xsd:complexType>
  
```

2649

2650 The following is the type of the row element contained in the result set. The value in the row are returned
 2651 in the same order as specified in the select clause of the query.

```

2652 <xsd:complexType name="tTaskQueryResultRow">
2653   <xsd:choice minOccurs="0" maxOccurs="unbounded">
2654     <xsd:element name="id" type="xsd:string"/>
2655     <xsd:element name="taskType" type="xsd:string"/>
2656     <xsd:element name="name" type="xsd:QName"/>
2657     <xsd:element name="status" type="tStatus"/>
2658     <xsd:element name="priority" type="htt:tPriority"/>
2659     <xsd:element name="taskInitiator"
2660       type="htt:tUser"/>
  
```

```

2661 <xsd:element name="taskStakeholders"
2662         type="htt:tOrganizationalEntity"/>
2663 <xsd:element name="potentialOwners"
2664         type="htt:tOrganizationalEntity"/>
2665 <xsd:element name="businessAdministrators"
2666         type="htt:tOrganizationalEntity"/>
2667 <xsd:element name="actualOwner" type="htt:tUser"/>
2668 <xsd:element name="notificationRecipients"
2669         type="htt:tOrganizationalEntity"/>
2670 <xsd:element name="createdTime" type="xsd:dateTime"/>
2671 <xsd:element name="createdBy" type="xsd:string"/>
2672 <xsd:element name="lastModifiedTime" type="xsd:dateTime"/>
2673 <xsd:element name="lastModifiedBy" type="xsd:string"/>
2674 <xsd:element name="activationTime" type="xsd:dateTime"/>
2675 <xsd:element name="expirationTime" type="xsd:dateTime"/>
2676 <xsd:element name="isSkipable" type="xsd:boolean"/>
2677 <xsd:element name="hasPotentialOwners" type="xsd:boolean"/>
2678 <xsd:element name="startByTime" type="xsd:dateTime"/>
2679 <xsd:element name="completeByTime" type="xsd:dateTime"/>
2680 <xsd:element name="presentationName" type="tPresentationName"/>
2681 <xsd:element name="presentationSubject"
2682         type="tPresentationSubject"/>
2683 <xsd:element name="renderingMethodName" type="xsd:QName"/>
2684 <xsd:element name="hasOutput" type="xsd:boolean"/>
2685 <xsd:element name="hasFault" type="xsd:boolean"/>
2686 <xsd:element name="hasAttachments" type="xsd:boolean"/>
2687 <xsd:element name="hasComments" type="xsd:boolean"/>
2688 <xsd:element name="escalated" type="xsd:boolean"/>
2689 <xsd:element name="parentTaskId" type="xsd:string"/>
2690 <xsd:element name="hasSubTasks" type="xsd:boolean"/>
2691 <xsd:element name="searchBy" type="xsd:string"/>
2692 <xsd:element name="outcome" type="xsd:string"/>
2693 <xsd:element name="taskOperations" type="tTaskOperations"/>
2694 <xsd:any namespace="##other" processContents="lax"/>
2695 </xsd:choice>
2696 </xsd:complexType>

```

2697 Complete Task View

2698 The table below is the set of columns used when defining select clause, where clause, and order-by
2699 clause of query operations. Conceptually, this set of columns defines a universal relation. As a result the
2700 query can be formulated without specifying a from clause. A WS-HumanTask Processor MAY extend this
2701 view by adding columns.

2702

Column Name	Type	Constraints
ID	xsd:string	
TaskType	Enumeration	Identifies the task type. The following values are allowed: <ul style="list-style-type: none"> • "TASK" for a human task • "NOTIFICATION" for notifications Note that notifications are simple tasks that do not block the progress of the caller,
Name	xsd:QName	
Status	Enumeration	For values see section 4.10 "Human Task Behavior and State Transitions"
Priority	htt:tPriority	
(GenericHumanRole)	xsd:tUser or htt:tOrganizationalEntity	
CreatedTime	xsd:dateTime	The time in UTC when the task has been created.
CreatedBy	xsd:string	
LastModifiedTime	xsd:dateTime	The time in UTC when the task has been last modified.
LastModifiedBy	xsd:string	
ActivationTime	xsd:dateTime	The time in UTC when the task has been activated.
ExpirationTime	xsd:dateTime	The time in UTC when the task will expire.
IsSkipable	xsd:boolean	
StartByTime	xsd:dateTime	The time in UTC when the task needs to be started. This time corresponds to the respective start deadline.
CompleteByTime	xsd:dateTime	The time in UTC when the task needs to be completed. This time corresponds to the respective end deadline.

Column Name	Type	Constraints
PresentationName	xsd:string	The task's presentation name.
PresentationSubject	xsd:string	The task's presentation subject.
RenderingMethodName	xsd:QName	The task's rendering method name.
HasOutput	xsd:boolean	
HasFault	xsd:boolean	
HasAttachments	xsd:boolean	
HasComments	xsd:boolean	
Escalated	xsd:boolean	
ParentTaskId	xsd:string	
HasSubTasks	xsd:boolean	
SearchBy	xsd:string	
Outcome	xsd:string	
TaskOperations	htt:tTaskOperations	

2704

2705 7.1.4 Administrative Operations

2706 The following operations are executed for administrative purposes.

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
activate	Activate the task, i.e. set the task to status <i>Ready</i> .	In task identifier Out void	Yes	Created	Ready
nominate	Nominate an organization entity to process the task. If it is nominated to one person then the new state of the task is <i>Reserved</i> . If it is nominated to several people then the new state of the task is	In task identifier organizational entity (htt:tOrganizationalEntity) Out void	Yes	Created	Ready Reserved

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
	<i>Ready.</i>				
setGenericHumanRole	Replace the organizational assignment to the task in one generic human role.	In task identifier generic human role organizational entity (htt:tOrganizationalEntity) Out void	Yes	Created Ready Reserved InProgress Suspended/Ready (from Ready) Suspended/Reserved (from Reserved) Suspended/InProgress (from InProgress)	(no state transition)

2707
2708

2709 **7.1.5 Operation Authorizations**

2710 The table below summarizes the required authorizations in terms of generic human roles to execute
 2711 participant, query and administrative operations. Thus, it is a precise definition of the generic human roles
 2712 as well. The sign plus ('+') means that the operation MUST be available for the generic human role. The
 2713 sign minus ('-') means that the operation MUST NOT be available for the generic human role. 'n/a'
 2714 indicates that the operation is not applicable and thus MUST NOT be available for the generic human
 2715 role. 'MAY' defines that vendor MAY chose to support the operation for the generic human role.

2716 If a person has multiple generic human roles on a human task or notification and she is allowed to
 2717 perform an operation in any of the roles then the invocation of the operation will not fail, otherwise
 2718 hta:illegalAccessFault and hta:recipientNotAllowed MUST be returned in the case of tasks
 2719 and notifications respectively. If a person is included in the list of excluded owners of a task then she
 2720 MUST NOT perform any of the operations.

2721 All batch operations (operations with a name prefix "batch") may be invoked by any caller; no specific
 2722 authorization is required. Missing authorizations for operations on individual tasks result in a report entry
 2723 in the batch operation's response message.

2724

Operation \ Role	Task Initiator	Task Stakeholders	Potential Owners	Actual Owner	Business Administrator	Notification Recipients
claim	-	MAY	+	n/a	MAY	n/a
start	-	MAY	+	+	MAY	n/a
stop	-	MAY	n/a	+	MAY	n/a
release	-	MAY	n/a	+	MAY	n/a
suspend	MAY	+	MAY	MAY	+	n/a
suspendUntil	MAY	+	MAY	MAY	+	n/a
resume	MAY	+	MAY	MAY	+	n/a
complete	-	MAY	n/a	+	MAY	n/a
remove	-	n/a	n/a	n/a	+	+
fail	-	MAY	n/a	+	MAY	n/a
setPriority	MAY	+	MAY	MAY	+	n/a
addAttachment	MAY	+	+	+	+	n/a
getAttachmentInfos	MAY	+	+	+	+	n/a
getAttachment	MAY	+	+	+	+	n/a
deleteAttachment	MAY	+	+	+	+	n/a
addComment	MAY	+	+	+	+	n/a
updateComment	MAY	+	+	+	+	n/a
deleteComment	MAY	+	+	+	+	n/a
getComments	MAY	+	+	+	+	n/a
skip	+	+	MAY	MAY	+	n/a
forward	MAY	+	MAY	+	+	n/a
delegate	MAY	+	MAY	+	+	n/a
getRendering	+	+	+	+	+	+
getRenderingTypes	+	+	+	+	+	+
getTaskDetails	MAY	+	+	+	+	+
getTaskDescription	+	+	+	+	+	+
getTaskOperations	+	+	+	+	+	+
setOutput	-	MAY	n/a	+	MAY	n/a
deleteOutput	-	MAY	n/a	+	MAY	n/a
setFault	-	MAY	n/a	+	MAY	n/a
deleteFault	-	MAY	n/a	+	MAY	n/a
getInput	+	+	+	+	+	n/a
getOutput	+	+	MAY	+	+	n/a
getFault	+	+	MAY	+	+	n/a
getOutcome	+	+	MAY	+	+	n/a
getTaskHistory	+	+	MAY	+	+	n/a
getTaskInstanceData	+	+	+	+	+	n/a
getSubtasks	+	+	+	+	+	n/a

Operation \ Role	Task Initiator	Task Stakeholders	Potential Owners	Actual Owner	Business Administrator	Notification Recipients
getSubtaskIdentifiers	+	+	+	+	+	n/a
hasSubtasks	+	+	+	+	+	n/a
getParentTask	+	+	MAY	+	+	n/a
getParentTaskIdentifier	+	+	MAY	+	+	n/a
isSubtask	+	+	+	+	+	n/a
instantiateSubTask	-	-	-	+	n/a	n/a
setTaskStartDeadlineExpression	MAY	+	-	-	+	n/a
setTaskStartDurationExpression	MAY	+	-	-	+	n/a
setTaskCompletionDeadlineExpression	MAY	+	-	-	+	n/a
setTaskCompletionDurationExpression	MAY	+	-	-	+	n/a
getMyTaskAbstracts	+	+	+	+	+	+
getMyTaskDetails	+	+	+	+	+	+
activate	+	+	n/a	n/a	+	-
nominate	MAY	-	-	-	+	-
setGenericHumanRole	-	-	-	-	+	-
batch*	+	+	+	+	+	+

2726

2727 7.2 XPath Extension Functions

2728 This section introduces XPath extension functions that are provided to be used within the definition of a
 2729 human task or notification. A WS-HumanTask Processor MUST support the XPath Functions listed below.
 2730 When defining properties using these XPath functions, note the initialization order in section 4.10.1.

2731 Definition of these XPath extension functions is provided in the table below. Input parameters that specify
 2732 task name, message part name or logicalPeopleGroup name MUST be literal strings. This restriction
 2733 does not apply to other parameters. Because XPath 1.0 functions do not support returning faults, an
 2734 empty node set is returned in the event of an error.

2735 XPath functions used for notifications in an escalation can access context from the enclosing task by
 2736 specifying that task's name.

2737

2738

2739
2740

Operation Name	Description	Parameters
getPotentialOwners	Returns the potential owners of the task. It MUST evaluate to an empty <code>htt:organizationalEntity</code> in case of an error. If the task name is not present the current task MUST be considered.	In <ul style="list-style-type: none"> task name (optional) Out <ul style="list-style-type: none"> potential owners (<code>htt:organizationalEntity</code>)
getActualOwner	Returns the actual owner of the task. It MUST evaluate to an empty <code>htt:user</code> in case there is no actual owner. If the task name is not present the current task MUST be considered.	In <ul style="list-style-type: none"> task name (optional) Out <ul style="list-style-type: none"> the actual owner (user id as <code>htt:user</code>)
getTaskInitiator	Returns the initiator of the task. It MUST evaluate to an empty <code>htt:user</code> in case there is no initiator. If the task name is not present the current task MUST be considered.	In <ul style="list-style-type: none"> task name (optional) Out <ul style="list-style-type: none"> the task initiator (user id as <code>htt:user</code>)
getTaskStakeholders	Returns the stakeholders of the task. It MUST evaluate to an empty <code>htt:organizationalEntity</code> in case of an error. If the task name is not present the current task MUST be considered.	In <ul style="list-style-type: none"> task name (optional) Out <ul style="list-style-type: none"> task stakeholders (<code>htt:organizationalEntity</code>)
getBusinessAdministrators	Returns the business administrators of the task. It MUST evaluate to an empty <code>htt:organizationalEntity</code> in case of an error. If the task name is not present the current task MUST be considered.	In <ul style="list-style-type: none"> task name (optional) Out <ul style="list-style-type: none"> business administrators (<code>htt:organizationalEntity</code>)
getExcludedOwners	Returns the excluded owners. It MUST evaluate to an empty <code>htt:organizationalEntity</code>	In <ul style="list-style-type: none"> task name (optional) Out

Operation Name	Description	Parameters
	<p>entity in case of an error.</p> <p>If the task name is not present the current task MUST be considered.</p>	<ul style="list-style-type: none"> excluded owners (htt:organizationalEntity)
getTaskPriority	<p>Returns the priority of the task.</p> <p>It MUST evaluate to "5" in case the priority is not explicitly set.</p> <p>If the task name is not present the current task MUST be considered.</p>	<p>In</p> <ul style="list-style-type: none"> task name (optional) <p>Out</p> <ul style="list-style-type: none"> priority (htt:tPriority)
getInput	<p>Returns the part of the task's input message.</p> <p>If the task name is not present the current task MUST be considered.</p>	<p>In</p> <ul style="list-style-type: none"> part name task name (optional) <p>Out</p> <ul style="list-style-type: none"> input message part
getSubtaskOutput	<p>Returns a node-set representing the specified part or contained elements of a sub task's output message. Only completed sub tasks of the current task MUST be considered</p>	<p>In</p> <ul style="list-style-type: none"> sub task name part name location path <p>Out</p> <ul style="list-style-type: none"> node-set of output message element(s)
getSubtaskOutputs	<p>Returns a node-set of simple-typed or complex-typed elements, constructed from the sub tasks' output documents in a routing pattern. The string parameter contains a location path evaluated on each sub task's output document. The individual node-sets are combined into the returned node-set. Only completed sub tasks of the current task MUST be considered</p>	<p>In</p> <ul style="list-style-type: none"> part name location path <p>Out</p> <ul style="list-style-type: none"> node-set of output message elements from sub tasks
getOutput	<p>Returns the part of the task's output message.</p> <p>If the task name is not present the current task MUST be considered</p>	<p>In</p> <ul style="list-style-type: none"> part name task name (optional) <p>Out</p>

Operation Name	Description	Parameters
		<ul style="list-style-type: none"> output message part
getCountOfSubTasks	Returns the number of sub tasks of a task If the task name is not present the current task MUST be considered	In <ul style="list-style-type: none"> task name (optional) Out <ul style="list-style-type: none"> Number of the task sub-tasks. If the task doesn't have sub tasks then 0 is returned
getCountOfFinishedSubTasks	Returns the number of finished sub tasks of a task If the task name is not present the current task MUST be considered	In <ul style="list-style-type: none"> task name (optional) Out <ul style="list-style-type: none"> Number of the finished task sub-tasks. If the task doesn't have sub tasks then 0 is returned
getCountOfSubTasksInState	Returns the number of a task subtasks that are in the specified state If the task name is not present the current task MUST be considered	In <ul style="list-style-type: none"> state task name (optional) Out <ul style="list-style-type: none"> Number of the task sub tasks in the specified state. If the task doesn't have sub tasks then 0 is returned
getCountOfSubTasksWithOutcome	Returns the number of a task sub tasks that match the given outcome If the task name is not present the current task MUST be considered	In <ul style="list-style-type: none"> outcome task name (optional) Out <ul style="list-style-type: none"> Number of the task sub tasks that match the specified outcome. If the task doesn't have sub tasks then 0 is returned
getLogicalPeopleGroup	Returns the value of a logical people group. In case of an error (e.g., when referencing a non existing logical people group) the <code>http:organizationalEntity</code> MUST contain an empty user list. If the task name is not present the current task MUST be considered.	In <ul style="list-style-type: none"> name of the logical people group The optional parameters that follow MUST appear in pairs. Each pair is defined as: <ul style="list-style-type: none"> the qualified name of a logical people group parameter the value for the named logical people group parameter; it can be an

Operation Name	Description	Parameters
		<p>XPath expression</p> <p>Out</p> <ul style="list-style-type: none"> the value of the logical people group (<code>htt:organizationalEntity</code>)
getOutcome	<p>Returns the outcome of the task. It MUST evaluate to an empty string in case there is no outcome specified for the task.</p> <p>If the task name is not present the current task MUST be considered.</p>	<p>In</p> <ul style="list-style-type: none"> task name (optional) <p>Out</p> <ul style="list-style-type: none"> the task outcome (<code>xsd:string</code>)
union	<p>Constructs an <code>organizationalEntity</code> containing every user that occurs in either set1 or set2, eliminating duplicate users.</p>	<p>In</p> <ul style="list-style-type: none"> set1 (<code>htt:organizationalEntity htt:user</code>) set2 (<code>htt:organizationalEntity htt:user</code>) <p>Out</p> <ul style="list-style-type: none"> result (<code>htt:organizationalEntity</code>)
intersect	<p>Constructs an <code>organizationalEntity</code> containing every user that occurs in both set1 and set2, eliminating duplicate users.</p>	<p>In</p> <ul style="list-style-type: none"> set1 (<code>htt:organizationalEntity htt:user</code>) set2 (<code>htt:organizationalEntity htt:user</code>) <p>Out</p> <ul style="list-style-type: none"> result (<code>htt:organizationalEntity</code>)
except	<p>Constructs an <code>organizationalEntity</code> containing every user that occurs in set1 but not in set2.</p> <p>Note: This function is required to allow enforcing the separation of duties</p>	<p>In</p> <ul style="list-style-type: none"> set1 (<code>htt:organizationalEntity htt:user</code>) set2 (<code>htt:organizationalEntity htt:user</code>) <p>Out</p>

Operation Name	Description	Parameters
	("4-eyes principle").	<ul style="list-style-type: none"> result (<code>http:organizationalEntity</code>)

2741

2742 In addition to the general-purpose functions listed above, the following aggregation functions MUST be
 2743 supported by a WS-HumanTask Processor. All aggregation functions take a node-set of strings,
 2744 booleans, or numbers as the first input parameter, and produce a result of the same type.

2745

Operation Name	Description	Parameters
concat	Returns the concatenation of all string nodes - returns an empty string for an empty node-set	In <ul style="list-style-type: none"> node-set of string nodes
concatWithDelimiter	Returns the concatenation of all string nodes, separated by the specified delimiter string - returns an empty string for an empty node-set	In <ul style="list-style-type: none"> node-set of string nodes delimiter string
leastFrequentOccurrence	Returns the least frequently occurring string value within all string nodes, or an empty string in case of a tie or for an empty node-set	In <ul style="list-style-type: none"> node-set of string nodes
mostFrequentOccurrence	Returns the most frequently occurring string value within all string nodes, or an empty string in case of a tie or for an empty node-set	In <ul style="list-style-type: none"> node-set of string nodes
voteOnString	Returns the most frequently occurring string value if its occurrence is above the specified percentage and there is no tie, or an empty string otherwise (including an empty node-set)	In <ul style="list-style-type: none"> node-set of string nodes percentage
and	Returns the conjunction of all boolean nodes - returns false for an empty node-set	In <ul style="list-style-type: none"> node-set of boolean nodes
or	Returns the disjunction of all boolean nodes - returns false for an empty node-set	In <ul style="list-style-type: none"> node-set of boolean nodes

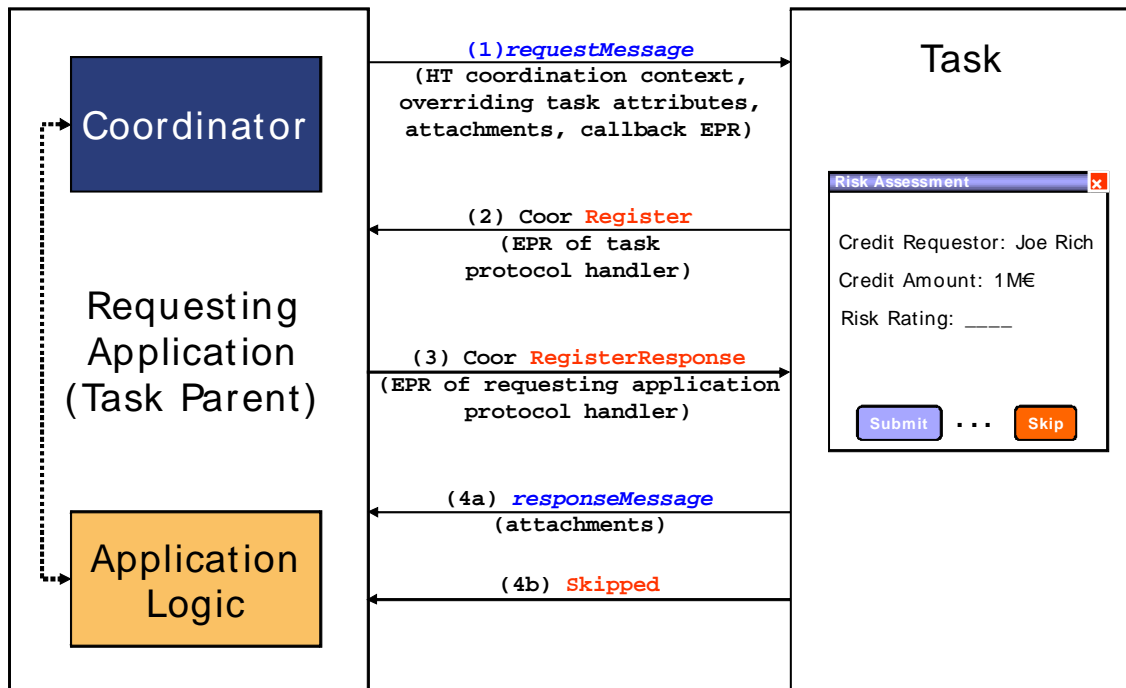
Operation Name	Description	Parameters
vote	Returns the most frequently occurring boolean value if its occurrence is above the specified percentage, or false otherwise (including an empty node-set)	In <ul style="list-style-type: none"> node-set of boolean nodes percentage
avg	Returns the average value of all number nodes - returns NaN for an empty node-set	In <ul style="list-style-type: none"> node-set of number nodes
max	Returns the maximum value of all number nodes - returns NaN for an empty node-set	In <ul style="list-style-type: none"> node-set of number nodes
min	Returns the minimum value of all number nodes - returns NaN for an empty node-set	In <ul style="list-style-type: none"> node-set of number nodes
sum	Returns the sum value of all number nodes - returns 0 for an empty node-set	In <ul style="list-style-type: none"> node-set of number nodes

2746

2747
2748
2749
2750
2751
2752
2753
2754
2755

8 Interoperable Protocol for Advanced Interaction with Human Tasks

Previous sections describe how to define standard invocable Web services that happen to be implemented by human tasks or notifications. Additional capability results from an application that is human task aware, and can control the autonomy and life cycle of the human tasks. To address this in an interoperable manner, a coordination protocol, namely the *WS-HumanTask coordination protocol*, is introduced to exchange life-cycle command messages between an application and an invoked human task. A simplified protocol applies to notifications.



2756

Figure 10: Message Exchange between Application and WS-HumanTask Processor

2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770

While we do not make any assumptions about the nature of the application in the following scenarios, in practice it would be hosted by an infrastructure that actually deals with the WS-HumanTask coordination protocol on the application's behalf.

In case of human tasks the following message exchanges are possible.

Scenario 1: At some point in time, the application invokes the human task through its service interface. In order to signal to the WS-HumanTask Processor that an instance of the human task can be created which is actually coordinated by the parent application, this request message contains certain control information. This control information consists of a coordination context of the WS-HumanTask coordination protocol, and optional human task attributes that are used to override aspects of the human task definition.

- The coordination context (see [WS-C] for more details on Web services coordination framework used here) contains the element `CoordinationType` that MUST specify the WS-HumanTask coordination type <http://docs.oasis-open.org/ns/bpel4people/ws-humantask/protocol/200803>. The inclusion of a coordination context within the request

2771 message indicates that the life cycle of the human tasks is managed via corresponding protocol
2772 messages from outside the WS-HumanTask Processor. The coordination context further contains
2773 in its `RegistrationService` element an endpoint reference that the WS-HumanTask
2774 Processor MUST use to register the task as a participant of that coordination type.
2775 Note: In a typical implementation, the parent application or its environment will create that
2776 coordination context by issuing an appropriate request against the WS-Coordination (WS-C)
2777 activation service, followed by registering the parent application as a `TaskParent` participant in
2778 that protocol.

- 2779 • The optional human task attributes allow overriding aspects of the definition of the human task
2780 from the calling application. The WS-HumanTask Parent MAY set values of the following
2781 attributes of the task definition:
 - 2782 ○ Priority of the task
 - 2783 ○ Actual people assignments for each of the generic human roles of the human task
 - 2784 ○ The skipable indicator which determines whether a task can actually be skipped at
2785 runtime.
 - 2786 ○ The amount of time by which the task activation is deferred.
 - 2787 ○ The expiration time for the human task after which the calling application is no longer
2788 interested in its result.

2789 After having created this request message, it is sent to the WS-HumanTask Processor (step (1) in Figure
2790 10). The WS-HumanTask Processor receiving that message MUST extract the coordination context and
2791 callback information, the human task attributes (if present) and the application payload. Before applying
2792 this application payload to the new human task, the WS-HumanTask Processor MUST register the human
2793 task to be created with the registration service passed as part of the coordination context (step (2) in
2794 Figure 10). The corresponding WS-C `Register` message MUST include the endpoint reference (EPR) of
2795 the protocol handler of the WS-HumanTask Processor that the WS-HumanTask Parent MUST use to
2796 send all protocol messages to WS-HumanTask Processor. This EPR is the value contained in the
2797 `ParticipantProtocolService` element of the `Register` message. Furthermore, the registration
2798 MUST be as a `HumanTask` participant by specifying the corresponding value in the
2799 `ProtocolIdentifier` element of the `Register` message. The WS-HumanTask Parent reacts to that
2800 message by sending back a `RegisterResponse` message. This message MUST contain in its
2801 `CoordinatorProtocolService` element the EPR of the protocol handler of the parent application,
2802 which MUST be used by the WS-HumanTask Processor for sending protocol messages to the parent
2803 application (step (3) in Figure 10).

2804 Now the instance of the human task is activated by the WS-HumanTask Processor, so the assigned
2805 person can perform the task (e.g. the risk assessment). Once the human task is successfully completed,
2806 a response message MUST be passed back to the parent application (step (4a) in Figure 10) by WS-
2807 HumanTask Processor.

2808 **Scenario 2:** If the human task is not completed with a result, but the assigned person determines that the
2809 task can be skipped (and hence reaches its *Obsolete* final state), then a “skipped” coordination protocol
2810 message MUST be sent from the WS-HumanTask Processor to its parent application (step (4b) in Figure
2811 10). No response message is passed back.

2812 **Scenario 3:** If the WS-HumanTask Parent needs to end prematurely before the invoked human task has
2813 been completed, it MUST send an `exit` coordination protocol message to the WS-HumanTask
2814 Processor causing the WS-HumanTask Processor to end its processing. A response message SHOULD
2815 NOT be passed back by WS-HumanTask Processor.

2816 In case of notifications to WS-HumanTask Processor, only some of the overriding attributes are
2817 propagated with the request message. Only priority and people assignments MAY be overridden for a
2818 notification, and the elements `isSkipable`, `expirationTime` and `attachments` MUST be ignored if present by
2819 WS-HumanTask Processor. Likewise, the WS-HumanTask coordination context, `attachments` and the
2820 callback EPR do not apply to notifications and MUST be ignored as well by WS-HumanTask Processor.
2821 Finally, a notification SHOULD NOT return WS-HumanTask coordination protocol messages. There
2822 SHOULD NOT be a message exchange beyond the initiating request message between the WS-
2823 HumanTask Processor and WS-HumanTask Parent.

2824 **8.1 Human Task Coordination Protocol Messages**

2825 The following section describes the behavior of the human task with respect to the protocol messages
 2826 exchanged with its requesting application which is human task aware. In particular, we describe which
 2827 state transitions trigger which protocol message and vice versa. WS-HumanTask Parent MUST support
 2828 WS-HumanTask Coordination protocol messages in addition to application requesting, responding and
 2829 fault messages.

2830 See diagram in section 4.10 “Human Task Behavior and State Transitions”.

- 2831 1. The initiating message containing a WS-HumanTask coordination context is received by the WS-
 2832 HumanTask Processor. This message MAY include ad hoc attachments that are to be made
 2833 available to the WS-HumanTask Processor. A new task is created. As part of the context, an EPR
 2834 of the registration service MUST be passed by WS-HumanTask Parent. This registration service
 2835 MUST be used by the hosting WS-HumanTask Processor to register the protocol handler
 2836 receiving the WS-HumanTask protocol messages sent by the requesting Application. If an error
 2837 occurs during the task instantiation the final state *Error* is reached and protocol message *fault*
 2838 MUST be sent to the requesting application by WS-HumanTask Processor.
- 2839 2. On successful completion of the task an application level response message MUST be sent and
 2840 the task moved to state *Completed*. When this happens, attachments created during the
 2841 processing of the task MAY be added to the response message. Attachments that had been
 2842 passed in the initiating message MUST NOT be returned. The response message outcome
 2843 MUST be set to the outcome of the task.
- 2844 3. On unsuccessful completion (completion with a fault message), an application level fault
 2845 message MUST be sent and the task moved to state *Failed*. When this happens, attachments
 2846 created during the processing of the task MAY be added to the response message. Attachments
 2847 that had been passed in the initiating message MUST NOT be returned.
- 2848 4. If the task experiences a non-recoverable error protocol message *fault* MUST be sent and
 2849 the task moved to state *Error*. Attachments MUST NOT be returned.
- 2850 5. If the task is skipable and is skipped then the WS-HumanTask Processor MUST send the
 2851 protocol message *skipped* and task MUST be moved to state *Obsolete*. Attachments MUST
 2852 NOT be returned.
- 2853 6. On receipt of protocol message *exit* the task MUST be moved to state *Exited*. This indicates
 2854 that the requesting application is no longer interested in any result produced by the task.

2855 The following table summarizes this behavior, the messages sent, and their direction, i.e., whether a
 2856 message is sent from the requesting application to the task (“out” in the column titled Direction) or vice
 2857 versa (“in”).

2858

Message	Direction	Human Task Behavior (and Protocol messages)
application request with WS-HT coordination context	in	Create task (Register)
application response	out	Successful completion with response
application fault response	out	Completion with fault response
htcp:Fault	out	Non-recoverable error
htcp:Exit	in	Requesting application is no longer interested in the task output
htcp:Skipped	out	Task moves to state Obsolete

2859 8.2 Protocol Messages

2860 All WS-HumanTask protocol messages have the following type:

```
2861 <xsd:complexType name="tProtocolMsgType">
2862   <xsd:sequence>
2863     <xsd:any namespace="##other" processContents="lax"
2864       minOccurs="0" maxOccurs="unbounded" />
2865   </xsd:sequence>
2866   <xsd:anyAttribute namespace="##other" processContents="lax" />
2867 </xsd:complexType>
```

2868 This message type is extensible and any implementation MAY use this extension mechanism to define
2869 proprietary attributes and content which are out of the scope of this specification.

2870 8.2.1 Protocol Messages Received by a Task Parent

2871 The following is the definition of the `htcp:skipped` message.

```
2872 <xsd:element name="skipped" type="htcp:tProtocolMsgType" />
2873 <wsdl:message name="skipped">
2874   <wsdl:part name="parameters" element="htcp:skipped" />
2875 </wsdl:message>
```

2876 The `htcp:skipped` message is used to inform the task parent (i.e. the requesting application) that the
2877 invoked task has been skipped. The task does not return any result.

2878 The following is the definition of the `htcp:fault` message.

```
2879 <xsd:element name="fault" type="htcp:tProtocolMsgType" />
2880 <wsdl:message name="fault">
2881   <wsdl:part name="parameters" element="htcp:fault" />
2882 </wsdl:message>
```

2883 The `htcp:fault` message is used to inform the task parent that the task has ended abnormally. The
2884 task does not return any result.

2885 8.2.2 Protocol Messages Received by a Task

2886 Upon receipt of the following `htcp:exit` message the task parent informs the task that it is no longer
2887 interested in its results.

```
2888 <xsd:element name="exit" type="htcp:tProtocolMsgType" />
2889 <wsdl:message name="exit">
2890   <wsdl:part name="parameters" element="htcp:exit" />
2891 </wsdl:message>
```

2892 8.3 WSDL of the Protocol Endpoints

2893 Protocol messages are received by protocol participants via operations of dedicated ports called protocol
2894 endpoints. In this section we specify the WSDL port types of the protocol endpoints needed to run the
2895 WS-HumanTask coordination protocol.

2896 8.3.1 Protocol Endpoint of the Task Parent

2897 An application that wants to create a task and wants to become a task parent MUST provide an endpoint
2898 implementing the following port type. This endpoint is the protocol endpoint of the task parent receiving
2899 protocol messages of the WS-HumanTask coordination protocol from a task. The operation used by the
2900 task to send a certain protocol message to the task parent is named by the message name of the protocol
2901 message concatenated by the string `Operation`. For example, the `skipped` message MUST be passed
2902 to the task parent by using the operation named `skippedOperation`.

```
2903 <wsdl:portType name="clientParticipantPortType">
```

```

2904 <wsdl:operation name="skippedOperation">
2905   <wsdl:input message="htcp:skipped" />
2906 </wsdl:operation>
2907 <wsdl:operation name="faultOperation">
2908   <wsdl:input message="htcp:fault" />
2909 </wsdl:operation>
2910 </wsdl:portType>

```

2911 8.3.2 Protocol Endpoint of the Task

2912 For a WS-HumanTask Definition a task MUST provide an endpoint implementing the following port type.
 2913 This endpoint is the protocol endpoint of the task receiving protocol messages of the WS-HumanTask
 2914 coordination protocol from a task parent. The operation used by the task parent to send a certain protocol
 2915 message to a task is named by the message name of the protocol message concatenated by the string
 2916 Operation. For example, the exit protocol message MUST be passed to the task by using the
 2917 operation named exitOperation.

```

2918 <wsdl:portType name="humanTaskParticipantPortType">
2919   <wsdl:operation name="exitOperation">
2920     <wsdl:input message="htcp:exit" />
2921   </wsdl:operation>
2922 </wsdl:portType>

```

2923 8.4 Providing Human Task Context

2924 The task context information is exchanged between the requesting application and a task or a notification.
 2925 In case of tasks, this information is passed as header fields of the request and response messages of the
 2926 task's operation. In case of notifications, this information is passed as header fields of the request
 2927 message of the notification's operation.

2928 8.4.1 SOAP Binding of Human Task Context

2929 In general, a SOAP binding specifies for message header fields how they are bound to SOAP headers. In
 2930 case of WS-HumanTask, the humanTaskRequestContext and humanTaskResponseContext
 2931 elements are simply mapped to SOAP header as a whole. The following listings show the SOAP binding
 2932 of the human task request context and human task response context in an infoset representation.

```

2933 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
2934   xmlns:htc="http://docs.oasis-open.org/ns/bpel4people/ws-
2935 humantask/context/200803">
2936   <S:Header>
2937     <htc:humanTaskRequestContext>
2938       <htc:priority>...</htc:priority>?
2939       <htc:attachments>...</htc:attachments>?
2940       <htc:peopleAssignments>...</htc:peopleAssignments>?
2941       <htc:isSkipable>...</htc:isSkipable>?
2942       <htc:activationDeferralTime>...</htc:activationDeferralTime>?
2943       <htc:expirationTime>...</htc:expirationTime>?
2944       ... extension elements ...
2945     </htc:humanTaskRequestContext>
2946   </S:Header>
2947   <S:Body>
2948     ...
2949   </S:Body>
2950 </S:Envelope>

```

```

2951
2952 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"

```

```

2953     xmlns:htc="http://docs.oasis-open.org/ns/bpel4people/ws-
2954 humantask/context/200803">
2955     <S:Header>
2956       <htc:humanTaskResponseContext>
2957         <htc:priority>...</htc:priority>?
2958         <htc:attachments>...</htc:attachments>?
2959         <htc:actualOwner>...</htc:actualOwner>?
2960         <htc:actualPeopleAssignments>...</htc:actualPeopleAssignments>?
2961         <htc:outcome>...</htc:outcome>?
2962         ... extension elements ...
2963       </htc:humanTaskResponseContext>
2964     </S:Header>
2965     <S:Body>
2966       ...
2967     </S:Body>
2968 </S:Envelope>

```

2969 The following listing is an example of a SOAP message containing a human task request context.

```

2970 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
2971     xmlns:htc="http://docs.oasis-open.org/ns/bpel4people/ws-
2972 humantask/context/200803">
2973   <S:Header>
2974     <htc:humanTaskRequestContext>
2975       <htc:priority>0</htc:priority>
2976       <htc:peopleAssignments>
2977         <htc:potentialOwners>
2978           <htt:organizationalEntity>
2979             <htt:user>Alan</htt:user>
2980             <htt:user>Dieter</htt:user>
2981             <htt:user>Frank</htt:user>
2982             <htt:user>Gerhard</htt:user>
2983             <htt:user>Ivana</htt:user>
2984             <htt:user>Karsten</htt:user>
2985             <htt:user>Matthias</htt:user>
2986             <htt:user>Patrick</htt:user>
2987           </htt:organizationalEntity>
2988         </htc:potentialOwners>
2989       </htc:peopleAssignments>
2990     </htc:humanTaskRequestContext>
2991   </S:Header>
2992   <S:Body>...</S:Body>
2993 </S:Envelope>

```

2994 8.4.2 Overriding Task Definition People Assignments

2995 The task context information exchanged contains a `potentialOwners` element, which can be used at
2996 task creation time to override the set of task assignments that we defined in the original task definition.
2997 Compliant implementations MUST allow overriding of simple tasks and routing patterns that are a single-
2998 level deep, i.e. routing patterns that don't have nested routing patterns. If the task context
2999 `potentialOwners` contains a list of `htt:user` and `htt:group`, and the task definition contains a
3000 routing pattern element `htt:parallel` or `htt:sequence` that has as its only children `htt:user` and
3001 `htt:group` elements, the WS-HumanTask Processor MUST replace the list in the task definition with the
3002 list in the task context. If the task definition contains only a list of `htt:user` and `htt:group`, then the
3003 WS-HumanTask Processor MUST replace the list of users from the task definition with the list of users in
3004 the task context.

3005 8.5 Human Task Policy Assertion

3006 In order to support discovery of Web services that support the human task contract that are available for
3007 coordination by another service, a *human task policy* assertion is defined by WS-HumanTask. This policy
3008 assertion can be associated with the business operation used by the invoking component (recall that the
3009 human task is restricted to have exactly one business operation). In doing so, the provider of a human
3010 task can signal whether or not the corresponding task can communicate with an invoking component via
3011 the WS-HumanTask coordination protocol.

3012 The following describes the policy assertion used to specify that an operation can be used to instantiate a
3013 human task with the proper protocol in place:

```
3014 <http:HumanTaskAssertion wsp:Optional="true"? ...>  
3015 ...  
3016 </http:HumanTaskAssertion>
```

3017 /http:HumanTaskAssertion

3018 This policy assertion specifies that the WS-HumanTask Parent, in this case the sender, **MUST**
3019 include context information for a human task coordination type passed with the message. The
3020 receiving human task **MUST** be instantiated with the WS-Human Task protocol in place by the
3021 WS-HumanTask Processor.

3022 /http:HumanTaskAssertion/@wsp:Optional="true"

3023 As defined in WS-Policy [WS-Policy], this is the compact notation for two policy alternatives, one
3024 with and one without the assertion. Presence of both policy alternatives indicates that the
3025 behavior indicated by the assertion is optional, such that a WS-HumanTask coordination context
3026 **MAY** be passed with an input message. If the context is passed the receiving human task **MUST**
3027 be instantiated with the WS-HumanTask protocol in place. The absence of the assertion is
3028 interpreted to mean that a WS-HumanTask coordination context **SHOULD NOT** be passed with
3029 an input message.

3030 The human task policy assertion indicates behavior for a single operation, thus the assertion has an
3031 Operation Policy Subject. WS-PolicyAttachment [WS-PolAtt] defines two policy attachment points with
3032 Operation Policy Subject, namely wsdl:portType/wsdl:operation and wsdl:binding/wsdl:operation.

3033 The <http:HumanTaskAssertion> policy assertion can also be used for notifications. In that case it
3034 means that the WS-HumanTask Parent, in this case the sender, **MAY** pass the human task context
3035 information with the message. Other headers, including headers with the coordination context are
3036 ignored.

3037 9 Task Parent Interactions with Lean Tasks

3038 9.1 Operations for Task Parent Applications

3039 A number of operations are involved in the life cycle of a lean task definition. These comprise:

- 3040 • Registering a lean task definition, such that it is available for later use
- 3041 • Unregistering a lean task definition, such that it is no longer available for later use
- 3042 • Listing lean task definitions, to determine what is available for use
- 3043 • Creating a lean task from a lean task definition

3044 An operation takes a well-defined set of parameters as its input. Passing an illegal parameter or an illegal
3045 number of parameters MUST result in the `htlt:illegalArgumentFault` being returned. Invoking an
3046 operation that is not allowed in the current state of the lean task definition MUST result in an
3047 `htlt:illegalStateFault`.

3048 By default, the identity of the person on behalf of which the operation is invoked is passed to the WS-
3049 HumanTask Processor. When the person is not authorized to perform the operation the
3050 `htlt:illegalAccessFault` MUST be returned.

3051 This specification does not stipulate the authentication, addressing, and binding scheme employed when
3052 calling an operation. This can be achieved using different mechanisms (e.g. WS-Security, WS-
3053 Addressing).

3054 9.2 Lean Task Interactions

3055 To enable lightweight task definition and creation by a WS-HumanTask Parent, a conformant WS-
3056 HumanTask Processor MUST provide the following operations:

- 3057 • `registerLeanTaskDefinition` API for registration
- 3058 • `unregisterLeanTaskDefinition` API for retraction
- 3059 • `listLeanTaskDefinitions` API for enumeration
- 3060 • `createLeanTask` and `createLeanTaskAsync` APIs for creation

3061 and invoke the following callback operation in response to `createLeanTaskAsync`:

- 3062 • `createLeanTaskAsyncCallback`

3063 9.2.1 Register a Lean Task Definition

```
3064 <xsd:element name="registerLeanTaskDefinition">
3065   <xsd:complexType>
3066     <xsd:sequence>
3067       <xsd:element name="taskDefinition" type="htd:tLeanTask" />
3068     </xsd:sequence>
3069   </xsd:complexType>
3070 </xsd:element>
3071 <xsd:element name="registerLeanTaskDefinitionResponse">
3072   <xsd:complexType>
3073     <xsd:sequence>
3074       <xsd:element name="taskName" type="xsd:NCName" />
3075     </xsd:sequence>
3076   </xsd:complexType>
3077 </xsd:element>
```


3078 The `htlt:registerLeanTaskDefinition` operation is used to create a new Lean Task definition that
3079 is available for future listing and consumption by the `htlt:listLeanTaskDefinitions` and
3080 `htlt:createLeanTask` / `htlt:createLeanTaskAsync` operations. If an existing Lean Task exists at
3081 the same name as the `htd:tLeanTask/@Name`, the WSHumanTask Processor SHOULD return an
3082 `htlt:illegalStateFault`.

3083 9.2.2 Unregister a Lean Task Definition

```
3084 <xsd:element name="unregisterLeanTaskDefinition">  
3085   <xsd:complexType>  
3086     <xsd:sequence>  
3087       <xsd:element name="taskName" type="xsd:NCName" />  
3088     </xsd:sequence>  
3089   </xsd:complexType>  
3090 </xsd:element>  
3091 <xsd:element name="unregisterLeanTaskDefinitionResponse">  
3092   <xsd:complexType>  
3093     <xsd:sequence>  
3094       <xsd:element name="taskName" type="xsd:NCName" />  
3095     </xsd:sequence>  
3096   </xsd:complexType>  
3097 </xsd:element>
```

3098 The `htlt:unregisterLeanTaskDefinition` operation is used to remove a Lean Task available for
3099 future listing and consumption by the `htlt:listLeanTaskDefinitions` and
3100 `htlt:createLeanTask` / `htlt:createLeanTaskAsync` operations. The WS-HumanTask Processor
3101 SHOULD also move any instances of lean tasks of this task definition to "Error" state. If the Lean Task
3102 does not already exist as a registered element, the WS-HumanTask Processor MUST return an
3103 `htlt:illegalArgumentFault`.

3104 9.2.3 List Lean Task Definitions

```
3105 <xsd:element name="listLeanTaskDefinitions">  
3106   <xsd:complexType>  
3107     <xsd:sequence>  
3108       <xsd:annotation>  
3109         <xsd:documentation>Empty message</xsd:documentation>  
3110       </xsd:annotation>  
3111     </xsd:sequence>  
3112   </xsd:complexType>  
3113 </xsd:element>  
3114 <xsd:element name="listLeanTaskDefinitionsResponse">  
3115   <xsd:complexType>  
3116     <xsd:sequence>  
3117       <xsd:element name="leanTaskDefinitions">  
3118         <xsd:complexType>  
3119           <xsd:sequence>  
3120             <xsd:element name="leanTaskDefinition" type="htd:tLeanTask"  
3121 minOccurs="0" maxOccurs="unbounded" />  
3122           </xsd:sequence>  
3123         </xsd:complexType>  
3124       </xsd:element>  
3125     </xsd:sequence>  
3126   </xsd:complexType>  
3127 </xsd:element>
```


3128 The `htlt:listLeanTaskDefinitions` operation is used to query the list of `htd:tLeanTask`
3129 elements that are registered Lean Tasks, as registered by the `htlt:registerLeanTaskDefinition`
3130 operation, and not subsequently unregistered by `htlt:unregisterLeanTaskDefinition`.

3131 9.2.4 Create a Lean Task

```
3132 <xsd:element name="CreateLeanTask">  
3133   <xsd:complexType>  
3134     <xsd:sequence>  
3135       <xsd:element name="inputMessage">  
3136         <xsd:complexType>  
3137           <xsd:sequence>  
3138             <xsd:any processContents="lax" namespace="##any" />  
3139           </xsd:sequence>  
3140         </xsd:complexType>  
3141       </xsd:element>  
3142       <xsd:element name="taskDefinition" type="htd:tLeanTask" minOccurs="0"/>  
3143       <xsd:element name="taskName" type="xsd:NCName" minOccurs="0" />  
3144     </xsd:sequence>  
3145   </xsd:complexType>  
3146 </xsd:element>  
3147 <xsd:element name="CreateLeanTaskResponse">  
3148   <xsd:complexType>  
3149     <xsd:sequence>  
3150       <xsd:element name="outputMessage">  
3151         <xsd:complexType>  
3152           <xsd:sequence>  
3153             <xsd:any processContents="lax" namespace="##any" />  
3154           </xsd:sequence>  
3155         </xsd:complexType>  
3156       </xsd:element>  
3157     </xsd:sequence>  
3158   </xsd:complexType>  
3159 </xsd:element>  
3160
```

3161 The `htlt:createLeanTask` operation is called by a WS-HumanTask Parent to create a task based on
3162 a Lean Task definition. This task definition either can be passed in directly to the operation or can
3163 reference a Lean Task definition previously sent via `htlt:registerLeanTaskDefinition`. These
3164 tasks follow the standard pattern of the Human Task Coordination protocol and is the operation on the
3165 portType used to create a task in that standard pattern, using the `humanTaskRequestContext` and
3166 `humanTaskResponseContext` as described in section 8.4.

3167 If both `taskName` and `taskDefinition` are set, the WS-HumanTask Processor MUST return an
3168 `htlt:illegalArgumentFault`. If `taskName` is set and a lean task has been registered by that name,
3169 the WS-HumanTask Process MUST use the registered lean task definition to create the task. If `taskName`
3170 is not set and a lean task has not been registered by that name, the WS-HumanTask Processor MUST
3171 return an `htlt:illegalArgumentFault`. If `taskDefinition` is set, the WS-HumanTask Processor MUST
3172 use the `taskDefinition` element as the type of the task to create. The WS-HumanTask Processor MUST
3173 use the `inputMessage` as the input message of the task and return the output message of the task in
3174 the `outputMessage` element.

3175 The `htlt:createLeanTask` operation is long-running because its execution includes the user
3176 interaction with the task owner. As a result, it is not meaningful to bind the request-response operation to
3177 a protocol that blocks any resources until the response is returned.

3178 Alternatively, instead of invoking the long-running request-response operation defined above, an
3179 interaction style using an asynchronous callback operation can be used. In this case, the WS-HumanTask
3180 Parent invokes the following `htlt:createLeanTaskAsync` operation and, as described in section 10,

3181 passes a WS-Addressing endpoint reference (EPR) in order to provide a callback address for delivering
3182 the lean task's output.

3183 Technically, `htlt:createLeanTaskAsync` is also a request-response operation in order to enable
3184 returning faults, but it returns immediately to the caller if the lean task is created successfully, without
3185 waiting for the lean task to complete.

```
3186 <xsd:element name="createLeanTaskAsync">  
3187   <xsd:complexType>  
3188     <xsd:sequence>  
3189       <xsd:element name="inputMessage">  
3190         <xsd:complexType>  
3191           <xsd:sequence>  
3192             <xsd:any processContents="lax" namespace="##any" />  
3193           </xsd:sequence>  
3194         </xsd:complexType>  
3195       </xsd:element>  
3196       <xsd:element name="taskDefinition" type="htd:tLeanTask" minOccurs="0"/>  
3197       <xsd:element name="taskName" type="xsd:NCName" minOccurs="0" />  
3198     </xsd:sequence>  
3199   </xsd:complexType>  
3200 </xsd:element>  
3201 <xsd:element name="createLeanTaskAsyncResponse">  
3202   <xsd:complexType>  
3203     <xsd:sequence/>  
3204   </xsd:complexType>  
3205 </xsd:element>
```

3206 Upon completion of the lean task, the WS-HumanTask Processor invokes the callback operation
3207 `htlt:createLeanTaskAsyncCallback` at the callback address specified in the EPR passed by the
3208 WS-HumanTask Parent.

```
3209 <xsd:element name="createLeanTaskAsyncCallback">  
3210   <xsd:complexType>  
3211     <xsd:sequence>  
3212       <xsd:element name="outputMessage">  
3213         <xsd:complexType>  
3214           <xsd:sequence>  
3215             <xsd:any processContents="lax" namespace="##any" />  
3216           </xsd:sequence>  
3217         </xsd:complexType>  
3218       </xsd:element>  
3219     </xsd:sequence>  
3220   </xsd:complexType>  
3221 </xsd:element>
```

3222 9.2.5 Endpoints for Lean Task Operations

3223 A WS-HumanTask Processor MUST provide an endpoint implementing the following port type. This
3224 endpoint is used to register, unregister, and list lean task definitions, and create a lean task given a
3225 particular definition and input message.

```
3226 <wsdl:portType name="leanTaskOperations">  
3227   <wsdl:operation name="registerLeanTaskDefinition">  
3228     <wsdl:input message="registerLeanTaskDefinition" />  
3229     <wsdl:output message="registerLeanTaskDefinitionResponse" />  
3230     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />  
3231     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />  
3232   </wsdl:operation>  
3233 </wsdl:portType>  
3234
```

```

3235 <wsdl:operation name="unregisterLeanTaskDefinition">
3236   <wsdl:input message="unregisterLeanTaskDefinition" />
3237   <wsdl:output message="unregisterLeanTaskDefinitionResponse" />
3238   <wsdl:fault name="illegalArgumentFault" message="illegalArgumentFault" />
3239   <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
3240 </wsdl:operation>
3241
3242 <wsdl:operation name="listLeanTaskDefinitions">
3243   <wsdl:input message="listLeanTaskDefinitions" />
3244   <wsdl:output message="listLeanTaskDefinitionsResponse" />
3245   <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
3246 </wsdl:operation>
3247
3248 <wsdl:operation name="createLeanTask">
3249   <wsdl:input message="createLeanTask" />
3250   <wsdl:output message="createLeanTaskResponse" />
3251   <wsdl:fault name="illegalArgumentFault" message="illegalArgumentFault" />
3252   <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
3253 </wsdl:operation>
3254
3255 <wsdl:operation name="createLeanTaskAsync">
3256   <wsdl:input message="createLeanTaskAsync" />
3257   <wsdl:output message="createLeanTaskAsyncResponse" />
3258   <wsdl:fault name="illegalArgumentFault" message="illegalArgumentFault" />
3259   <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
3260 </wsdl:operation>
3261
3262 </wsdl:portType>

```

3263 A WS-HumanTask Parent invoking the `htlt:createLeanTaskAsync` operation MUST provide an
3264 endpoint implementing the following callback port type.

```

3265 <wsdl:portType name="leanTaskCallbackOperations">
3266
3267   <wsdl:operation name="createLeanTaskAsyncCallback">
3268     <wsdl:input message="createLeanTaskAsyncCallback" />
3269   </wsdl:operation>
3270
3271 </wsdl:portType>

```

3272

10 Providing Callback Information for Human Tasks

3273 WS-HumanTask extends the information model of a WS-Addressing endpoint reference (EPR) defined in
3274 [WS-Addr-Core] (see [WS-Addr-SOAP] and [WS-Addr-WSDL] for more details). This extension is needed
3275 to support passing information to human tasks about ports and operations of a caller receiving responses
3276 from such human tasks.

3277 Passing this callback information from a WS-HumanTask Parent (i.e. a requesting application) to the WS-
3278 HumanTask Processor MAY override static deployment information that may have been set.

10.1 EPR Information Model Extension

3280 Besides the properties of an endpoint reference (EPR) defined by [WS-Addr-Core] WS-HumanTask
3281 defines the following abstract properties:

3282 [response action] : xsd:anyURI (0..1)

3283 This property contains the value of the [action] message addressing property to be sent within the
3284 response message.

3285 [response operation] : xsd:NCName (0..1)

3286 This property contains the name of a WSDL operation.

3287 Each of these properties is a child element of the [metadata] property of an endpoint reference. An
3288 endpoint reference passed by a caller to a WS-HumanTask Processor MUST contain the [metadata]
3289 property. Furthermore, this [metadata] property MUST contain either a [response action] property or a
3290 [response operation] property.

3291 If present, the value of the [response action] property MUST be used by the WS-HumanTask Processor
3292 hosting the responding human task to specify the value of the [action] message addressing property of
3293 the response message sent back to the caller. Furthermore, the [destination] property of this response
3294 message MUST be copied from the [address] property of the EPR contained in the original request
3295 message by the WS-HumanTask Processor.

3296 If present, the value of the [response operation] property MUST be the name of an operation of the port
3297 type implemented by the endpoint denoted by the [address] property of the EPR. The corresponding port
3298 type MUST be included as a WSDL 1.1 definition nested within the [metadata] property of the EPR (see
3299 [WS-Addr-WSDL]). The WS-HumanTask Processor hosting the responding human task MUST use the
3300 value of the [response operation] property as operation of the specified port type at the specified endpoint
3301 to send the response message. Furthermore, the [metadata] property MUST contain WSDL 1.1 binding
3302 information corresponding to the port type implemented by the endpoint denoted by the [address]
3303 property of the EPR.

3304 The EPR sent from the caller to the WS-HumanTask Processor MUST identify the instance of the caller.
3305 This MUST be done by the caller in one of the two ways: First, the value of the [address] property can
3306 contain a URL with appropriate parameters uniquely identifying the caller instance. Second, appropriate
3307 [reference parameters] properties are specified within the EPR. The values of these [reference
3308 parameters] uniquely identify the caller within the scope of the URI passed within the [address] property.

10.2 XML Infoset Representation

3309 The following describes the infoset representation of the EPR extensions introduced by WS-HumanTask:

```
3311 <wsa:EndpointReference>  
3312   <wsa:Address>xsd:anyURI</wsa:Address>  
3313   <wsa:ReferenceParameters>xsd:any*</wsa:ReferenceParameters>?  
3314   <wsa:Metadata>  
3315     <htcp:responseAction>xsd:anyURI</htcp:responseAction>?  
3316     <htcp:responseOperation>xsd:NCName</htcp:responseOperation>?  
3317   </wsa:Metadata>
```

3318 `</wsa:EndpointReference>`

3319 `/wsa:EndpointReference/wsa:Metadata`

3320 This element of the EPR MUST be sent by WS-HumanTask Parent, the caller, to the WS-
3321 HumanTask Processor . It MUST either contain WSDL 1.1 metadata specifying the information to
3322 access the endpoint (i.e. its port type, bindings or ports) according to [WS-Addr-WSDL] as well as
3323 a `<htcp:responseOperation>` element, or it MUST contain a `<htcp:responseAction>`
3324 element.

3325 `/wsa:EndpointReference/wsa:Metadata/htcp:responseAction`

3326 This element (of type `xsd:anyURI`) specifies the value of the [action] message addressing
3327 property to be used by the receiving WS-HumanTask Processor when sending the response
3328 message from the WS-HumanTask Processor back to the caller. If this element is specified the
3329 `<htcp:responseOperation>` element MUST NOT be specified by the caller.

3330 `/wsa:EndpointReference/wsa:Metadata/htcp:responseOperation`

3331 This element (of type `xsd:NCName`) specifies the name of the operation that MUST be used by
3332 the receiving WS-HumanTask Processor to send the response message from the WS-
3333 HumanTask Processor back to the caller.. If this element is specified the
3334 `<htcp:responseAction>` element MUST NOT be specified by the WS-HumanTask Parent.

3335 Effectively, WS-HumanTask defines two ways to pass callback information from the caller to the human
3336 task. First, the EPR contains just the value of the [action] message addressing property that MUST be
3337 used by the WS-HumanTask Processor within the response message (i.e. the
3338 `<htcp:responseAction>` element). Second, the EPR contains the WSDL 1.1 metadata for the port
3339 receiving the response operation. In this case, for the callback information the WS-HumanTask Parent
3340 MUST specify which operation of that port is to be used (i.e. the `<htcp:responseOperation>`
3341 element). In both cases, the response is typically sent to the address specified in the `<wsa:Address>`
3342 element of the EPR contained in the original request message; note, that [WS-Addr-WSDL] does not
3343 exclude redirection to other addresses than the one specified, but the corresponding mechanisms are out
3344 of the scope of the specification.

3345 The following example of an endpoint reference shows the usage of the `<htcp:responseAction>`
3346 element. The `<wsa:Metadata>` elements contain the `<htcp:responseAction>` element that
3347 specifies the value of the [action] message addressing property to be used by the WS-HumanTask
3348 Processor when sending the response message back to the caller. This value is
3349 `http://example.com/LoanApproval/approvalResponse`. The value of the [destination] message
3350 addressing property to be used is given in the `<wsa:Address>` element, namely
3351 `http://example.com/LoanApproval/loan?ID=42`. Note that this URL includes the HTTP search
3352 part with the parameter `ID=42` which uniquely identifies the instance of the caller.

```
3353 <wsa:EndpointReference
3354   xmlns:wsa="http://www.w3.org/2005/08/addressing">
3355
3356   <wsa:Address>http://example.com/LoanApproval/loan?ID=42</wsa:Address>
3357
3358   <wsa:Metadata>
3359     <htcp:responseAction>
3360       http://example.com/LoanApproval/approvalResponse
3361     </htcp:responseAction>
3362   </wsa:Metadata>
3363
3364 </wsa:EndpointReference>
```

3365 The following example of an endpoint reference shows the usage of the `<htcp:responseOperation>`
3366 element and corresponding WSDL 1.1 metadata. The port type of the caller that receives the response
3367 message from the WS-HumanTask Processor is defined using the `<wsdl:portType>` element. In our
3368 example it is the `LoanApprovalPT` port type. The definition of the port type is nested in a corresponding
3369 WSDL 1.1 `<wsdl:definitions>` element in the `<wsa:Metadata>` element. This

3370 <wsdl:definitions> element also contains a binding for this port type as well as a corresponding
3371 port definition nested in a <wsdl:service> element. The <http:responseOperation> element
3372 specifies that the approvalResponse operation of the LoanApprovalPT port type is used to send the
3373 response to the caller. The address of the actual port to be used which implements the
3374 LoanApprovalPT port type and thus the approvalResponse operation is given in the
3375 <wsa:Address> element, namely the URL http://example.com/LoanApproval/loan. The
3376 unique identifier of the instance of the caller is specified in the <xmp:MyInstanceID> element nested in
3377 the <wsa:ReferenceParameters> element.

```
3378 <wsa:EndpointReference
3379   xmlns:wsa="http://www.w3.org/2005/08/addressing">
3380
3381   <wsa:Address>http://example.com/LoanApproval/loan</wsa:Address>
3382
3383   <wsa:ReferenceParameters>
3384     <xmp:MyInstanceID>42</xmp:MyInstanceID>
3385   </wsa:ReferenceParameters>
3386
3387   <wsa:Metadata>
3388
3389     <wsdl:definitions ...>
3390
3391     <wsdl:portType name="LoanApprovalPT">
3392       <wsdl:operation name="approvalResponse">...</wsdl:operation>
3393       ...
3394     </wsdl:portType>
3395
3396     <wsdl:binding name="LoanApprovalSoap" type="LoanApprovalPT">
3397       ...
3398     </wsdl:binding>
3399
3400     <wsdl:service name="LoanApprovalService">
3401       <wsdl:port name="LA" binding="LoanApprovalSoap">
3402         <soap:address
3403           location="http://example.com/LoanApproval/loan" />
3404         </wsdl:port>
3405         ...
3406       </wsdl:service>
3407
3408     </wsdl:definitions>
3409
3410     <http:responseOperation>approvalResponse</http:responseOperation>
3411
3412   </wsa:Metadata>
3413
3414 </wsa:EndpointReference>
```

3415 **10.3 Message Addressing Properties**

3416 Message addressing properties provide references for the endpoints involved in an interaction at the
3417 message level. For this case, WS-HumanTask Processor uses the message addressing properties
3418 defined in [WS-Addr-Core] for the request message as well as for the response message.

3419 The request message sent by the caller (i.e. the requesting application) to the human task uses the
3420 message addressing properties as described in [WS-Addr-Core]. WS-HumanTask refines the use of the
3421 following message addressing properties:

- 3422 • The [reply endpoint] message addressing property MUST contain the EPR to be used by the WS-
3423 HumanTask Processor to send its response to.

3424 Note that the [fault endpoint] property MUST NOT be used by WS-HumanTask Processor. This is
3425 because via one-way operation no application level faults are returned to the caller.

3426 The response message sent by the WS-HumanTask Processor to the caller uses the message
3427 addressing properties as defined in [WS-Addr-Core] and refines the use of the following properties:

- 3428 • The value of the [action] message addressing property is set as follows:
 - 3429 • If the original request message contains the <htcp:responseAction> element in the
3430 <wsa:Metadata> element of the EPR of the [reply endpoint] message addressing property,
3431 the value of the former element MUST be copied into the [action] property of the response
3432 message by WS-HumanTask Processor.
 - 3433 • If the original request message contains the <htcp:responseOperation> element (and,
3434 thus, WSDL 1.1 metadata) in the <wsa:Metadata> element of the EPR of the [reply
3435 endpoint] message addressing property, the value of the [action] message addressing
3436 property of the response message is determined as follows:
 - 3437 • Assume that the WSDL 1.1 metadata specifies within the binding chosen a value for the
3438 soapaction attribute on the soap:operation element of the response operation.
3439 Then, this value MUST be used as value of the [action] property by WS-HumanTask
3440 Processor.
 - 3441 • If no such soapaction attribute is provided, the value of the [action] property MUST be
3442 derived as specified in [WS-Addr-WSDL] by WS-HumanTask Processor.
- 3443 • Reference parameters are mapped as specified in [WS-Addr-SOAP].

3444 10.4 SOAP Binding

3445 A SOAP binding specifies how abstract message addressing properties are bound to SOAP headers. In
3446 this case, WS-HumanTask Processor MUST use the mappings as specified by [WS-Addr-SOAP].

3447 The following is an example of a request message sent from the caller to the WS-HumanTask Processor
3448 containing the <htcp:responseAction> element in the incoming EPR. The EPR is mapped to SOAP
3449 header fields as follows: The endpoint reference to be used by the human task for submitting its response
3450 message to is contained in the <wsa:ReplyTo> element. The address of the endpoint is contained in the
3451 <wsa:Address> element. The identifier of the instance of the caller to be encoded as reference
3452 parameters in the response message is nested in the <wsa:ReferenceParameters> element. The
3453 value of the <wsa:Action> element to be set by the human task in its response to the caller is in the
3454 <htcp:responseAction> element nested in the <wsa:Metadata> element of the EPR.

```
3455 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"  
3456   xmlns:wsa="http://www.w3.org/2005/08/addressing"  
3457   xmlns:htcp="http://docs.oasis-open.org/ns/bpel4people/ws-  
3458   humantask/protocol/200803">  
3459  
3460   <S:Header>  
3461     <wsa:ReplyTo>  
3462       <wsa:Address>http://example.com/LoanApproval/loan</wsa:Address>  
3463       <wsa:ReferenceParameters>  
3464         <xmp:MyInstanceID>42</xmp:MyInstanceID>  
3465       </wsa:ReferenceParameters>  
3466       <wsa:Metadata>  
3467         <htcp:responseAction>  
3468           http://example.com/LoanApproval/approvalResponse  
3469         </htcp:responseAction>  
3470       </wsa:Metadata>  
3471     </wsa:ReplyTo>  
3472   </S:Header>  
3473  
3474   <S:Body>...</S:Body>
```

3475 </S:Envelope>

3476 The following is an example of a response message corresponding to the request message discussed
3477 above. This response is sent from the WS-HumanTask Processor back to the caller. The <wsa:To>
3478 element contains a copy of the <wsa:Address> element of the original request message. The
3479 <wsa:Action> element is copied from the <htcp:responseAction> element of the original request
3480 message. The reference parameters are copied as standalone elements (the <xmp:MyInstanceID>
3481 element below) out of the <wsa:ReferenceParameters> element of the request message.

```
3482 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope "  
3483   xmlns:wsa="http://www.w3.org/2005/08/addressing" >  
3484   <S:Header>  
3485     <wsa:To>  
3486       <wsa:Address>http://example.com/LoanApproval/loan</wsa:Address>  
3487     </wsa:To>  
3488     <wsa:Action>  
3489       http://example.com/LoanApproval/approvalResponse  
3490     </wsa:Action>  
3491     <xmp:MyInstanceID wsa:IsReferenceParameter='true' >  
3492       42  
3493     </xmp:MyInstanceID>  
3494   </S:Header>  
3495   <S:Body>...</S:Body>  
3496 </S:Envelope>
```

3497 The following is an example of a request message sent from the caller to the WS-HumanTask Processor
3498 containing the <htcp:responseOperation> element and corresponding WSDL metadata in the
3499 incoming EPR. The EPR is mapped to SOAP header fields as follows: The endpoint reference to be used
3500 by the WS-HumanTask Processor for submitting its response message to is contained in the
3501 <wsa:ReplyTo> element. The address of the endpoint is contained in the <wsa:Address> element.
3502 The identifier of the instance of the caller to be encoded as reference parameters in the response
3503 message is nested in the <wsa:ReferenceParameters> element. The WSDL metadata of the
3504 endpoint is contained in the <wsdl:definitions> element. The name of the operation of the endpoint
3505 to be used to send the response message to is contained in the <htcp:responseOperation>
3506 element. Both elements are nested in the <wsa:Metadata> element of the EPR. These elements
3507 provide the basis to determine the value of the action header field to be set by the WS-HumanTask
3508 Processor in its response to the caller.

```
3509 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope "  
3510   xmlns:wsa="http://www.w3.org/2005/08/addressing"  
3511   xmlns:htcp="http://docs.oasis-open.org/ns/bpel4people/ws-  
3512   humantask/protocol/200803" >  
3513   <S:Header>  
3514     <wsa:ReplyTo>  
3515       <wsa:Address>http://example.com/LoanApproval/loan</wsa:Address>  
3516     </wsa:ReplyTo>  
3517     <wsa:ReferenceParameters>  
3518       <xmp:MyInstanceID>42</xmp:MyInstanceID>  
3519     </wsa:ReferenceParameters>  
3520     <wsa:Metadata>  
3521       <wsdl:definitions  
3522         targetNamespace="http://example.com/loanApproval "  
3523         xmlns:wsdl="..." xmlns:soap="..." >  
3524         <wsdl:portType name="LoanApprovalPT" >  
3525           <wsdl:operation name="approvalResponse" >  
3526             <wsdl:input name="approvalInput" ... />  
3527         </wsdl:portType >  
3528       </wsdl:definitions >  
3529     </wsa:Metadata >  
3530   </S:Header >  
3531   <S:Body >
```



```

3531     </wsdl:operation>
3532     ...
3533 </wsdl:portType>
3534
3535     <wsdl:binding name="LoanApprovalSoap"
3536         type="LoanApprovalPT">
3537         ...
3538     </wsdl:binding>
3539
3540     <wsdl:service name="LoanApprovalService">
3541         <wsdl:port name="LA" binding="LoanApprovalSoap">
3542             <soap:address
3543                 location="http://example.com/LoanApproval/loan" />
3544         </wsdl:port>
3545         ...
3546     </wsdl:service>
3547 </wsdl:definitions>
3548
3549     <htcp:responseOperation>
3550         approvalResponse
3551     </htcp:responseOperation>
3552
3553     </wsa:Metadata>
3554 </wsa:ReplyTo>
3555
3556 </S:Header>
3557 <S:Body>...</S:Body>
3558 </S:Envelope>

```

3559 The following is an example of a response message corresponding to the request message before; this
3560 response is sent from the WS-HumanTask Processor back to the caller. The `<wsa:To>` element contains
3561 a copy of the `<wsa:Address>` field of the original request message. The reference parameters are
3562 copied as standalone element (the `<xmp:MyInstanceID>` element below) out of the
3563 `<htcp:ReferenceParameters>` element of the request message. The value of the `<wsa:Action>`
3564 element is composed according to [WS-Addr-WSDL] from the target namespace, port type name, name
3565 of the response operation to be used, and name of the input message of this operation given in the code
3566 snippet above.

```

3567 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
3568     xmlns:wsa="http://www.w3.org/2005/08/addressing"
3569     xmlns:htd="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803">
3570 <S:Header>
3571     <wsa:To>http://example.com/LoanApproval/loan</wsa:To>
3572     <wsa:Action>
3573         http://example.com/loanApproval/...
3574         ...LoanApprovalPT/approvalResponse/ApprovalInput
3575     </wsa:Action>
3576     <xmp:MyInstanceID wsa:IsReferenceParameter='true'>
3577         42
3578     </xmp:MyInstanceID>
3579 </S:Header>
3580 <S:Body>...</S:Body>
3581 </S:Envelope>

```

3582

11 Security Considerations

3583 WS-HumanTask does not mandate the use of any specific mechanism or technology for client
3584 authentication. However, a client **MUST** provide a principal or the principal **MUST** be obtainable by the
3585 WS-HumanTask Processor.

3586 When using task APIs via SOAP bindings, compliance with the WS-I Basic Security Profile 1.0 is
3587 **RECOMMENDED**.

3588

12 Conformance

3589 The XML schema pointed to by the RDDL document at the namespace URI, defined by this specification,
3590 are considered to be authoritative and take precedence over the XML schema defined in the appendix of
3591 this document.

3592 There are four conformance targets defined as part of this specification: a WS-HumanTask Definition, a
3593 WS-HumanTask Processor, a WS-HumanTask Parent and a WS-HumanTask Client (see section 2.3). In
3594 order to claim conformance with WS-HumanTask 1.1, the conformance targetes **MUST** comply with all
3595 normative statements in this specification, notably all **MUST** statements have to be implemented.

3596 A. Portability and Interoperability Considerations

3597 This section illustrates the portability and interoperability aspects addressed by WS-HumanTask:

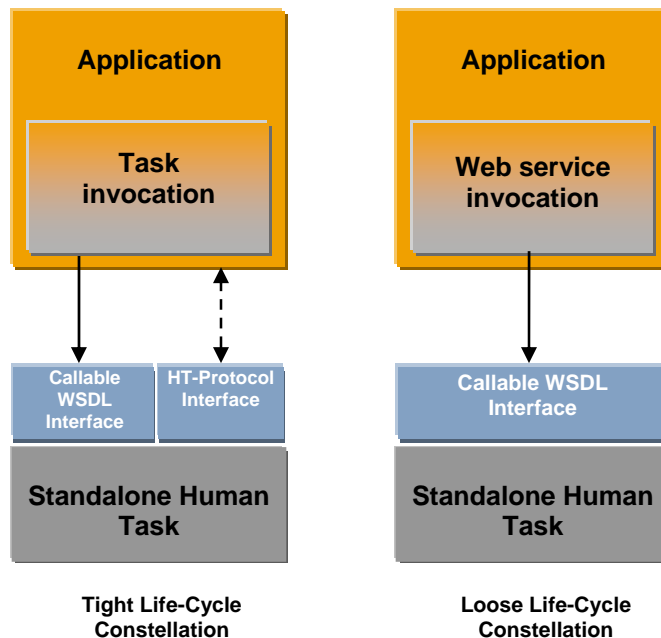
- 3598 • Portability - The ability to take human tasks and notifications created in one vendor's environment
3599 and use them in another vendor's environment.
- 3600 • Interoperability - The capability for multiple components (task infrastructure, task list clients and
3601 applications or processes with human interactions) to interact using well-defined messages and
3602 protocols. This enables combining components from different vendors allowing seamless
3603 execution.

3604 Portability requires support of WS-HumanTask artifacts.

3605 Interoperability between task infrastructure and task list clients is achieved using the operations for client
3606 applications.

3607 Interoperability between applications and task infrastructure from different vendors subsumes two
3608 alternative constellations depending on how tightly the life-cycles of the task and the invoking
3609 application are coupled with each other. This is shown in the figure below:

3610 Tight Life-Cycle Constellation: Applications are human task aware and control the life cycle of tasks.
3611 Interoperability between applications and WS-HumanTask Processors is achieved using the WS-
3612 HumanTask coordination protocol.



3613 Loose Life-Cycle Constellation: Applications use basic Web services protocols to invoke Web services
3614 implemented as human tasks. In this case standard Web services interoperability is achieved and
3615 applications do not control the life cycle of tasks.

3616

B. WS-HumanTask Language Schema

```
3617 <?xml version="1.0" encoding="UTF-8"?>
3618 <!--
3619   Copyright (c) OASIS Open 2009. All Rights Reserved.
3620 -->
3621 <xsd:schema
3622   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3623   xmlns="http://docs.oasis-open.org/ns/bpel4people/ws-humanTask/200803"
3624   targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
3625   humanTask/200803"
3626   elementFormDefault="qualified" blockDefault="#all">
3627
3628   <xsd:annotation>
3629     <xsd:documentation>
3630       XML Schema for WS-HumanTask 1.1 - WS-HumanTask Task Definition Language
3631     </xsd:documentation>
3632   </xsd:annotation>
3633
3634   <!-- other namespaces -->
3635   <xsd:import namespace="http://www.w3.org/XML/1998/namespace"
3636     schemaLocation="http://www.w3.org/2001/xml.xsd" />
3637
3638   <!-- base types for extensible elements -->
3639   <xsd:complexType name="tExtensibleElements">
3640     <xsd:sequence>
3641       <xsd:element name="documentation" type="tDocumentation" minOccurs="0"
3642   maxOccurs="unbounded" />
3643       <xsd:any namespace="##other" processContents="lax" minOccurs="0"
3644   maxOccurs="unbounded" />
3645     </xsd:sequence>
3646     <xsd:anyAttribute namespace="##other" processContents="lax" />
3647   </xsd:complexType>
3648
3649   <xsd:complexType name="tDocumentation" mixed="true">
3650     <xsd:sequence>
3651       <xsd:any namespace="##other" processContents="lax" minOccurs="0"
3652   maxOccurs="unbounded" />
3653     </xsd:sequence>
3654     <xsd:attribute ref="xml:lang" />
3655   </xsd:complexType>
3656
3657   <xsd:complexType name="tExtensibleMixedContentElements"
3658     mixed="true">
3659     <xsd:sequence>
3660       <xsd:element name="documentation" type="tDocumentation" minOccurs="0"
3661   maxOccurs="unbounded" />
3662       <xsd:any namespace="##other" processContents="lax" minOccurs="0"
3663   maxOccurs="unbounded" />
3664     </xsd:sequence>
3665     <xsd:anyAttribute namespace="##other" processContents="lax" />
3666   </xsd:complexType>
3667
3668   <!-- human interactions definition -->
3669   <xsd:element name="humanInteractions" type="tHumanInteractions" />
3670   <xsd:complexType name="tHumanInteractions">
```

```

3671     <xsd:complexContent>
3672         <xsd:extension base="tExtensibleElements">
3673             <xsd:sequence>
3674                 <xsd:element name="extensions" type="tExtensions" minOccurs="0" />
3675                 <xsd:element name="import" type="tImport" minOccurs="0"
3676 maxOccurs="unbounded" />
3677                 <xsd:element name="logicalPeopleGroups" type="tLogicalPeopleGroups"
3678 minOccurs="0" />
3679                 <xsd:element name="tasks" type="tTasks" minOccurs="0" />
3680                 <xsd:element name="notifications" type="tNotifications"
3681 minOccurs="0" />
3682             </xsd:sequence>
3683             <xsd:attribute name="targetNamespace" type="xsd:anyURI"
3684 use="required" />
3685             <xsd:attribute name="queryLanguage" type="xsd:anyURI" />
3686             <xsd:attribute name="expressionLanguage" type="xsd:anyURI" />
3687         </xsd:extension>
3688     </xsd:complexContent>
3689 </xsd:complexType>
3690
3691 <xsd:complexType name="tExtensions">
3692     <xsd:complexContent>
3693         <xsd:extension base="tExtensibleElements">
3694             <xsd:sequence>
3695                 <xsd:element name="extension" type="tExtension"
3696 maxOccurs="unbounded" />
3697             </xsd:sequence>
3698         </xsd:extension>
3699     </xsd:complexContent>
3700 </xsd:complexType>
3701
3702 <xsd:complexType name="tExtension">
3703     <xsd:complexContent>
3704         <xsd:extension base="tExtensibleElements">
3705             <xsd:attribute name="namespace" type="xsd:anyURI" use="required" />
3706             <xsd:attribute name="mustUnderstand" type="tBoolean" use="required"
3707 />
3708         </xsd:extension>
3709     </xsd:complexContent>
3710 </xsd:complexType>
3711
3712 <xsd:element name="import" type="tImport" />
3713 <xsd:complexType name="tImport">
3714     <xsd:complexContent>
3715         <xsd:extension base="tExtensibleElements">
3716             <xsd:attribute name="namespace" type="xsd:anyURI" use="optional" />
3717             <xsd:attribute name="location" type="xsd:anyURI" use="optional" />
3718             <xsd:attribute name="importType" type="xsd:anyURI" use="required" />
3719         </xsd:extension>
3720     </xsd:complexContent>
3721 </xsd:complexType>
3722
3723 <xsd:element name="logicalPeopleGroups" type="tLogicalPeopleGroups" />
3724 <xsd:complexType name="tLogicalPeopleGroups">
3725     <xsd:complexContent>
3726         <xsd:extension base="tExtensibleElements">
3727             <xsd:sequence>

```

```

3728         <xsd:element name="logicalPeopleGroup" type="tLogicalPeopleGroup"
3729 maxOccurs="unbounded" />
3730     </xsd:sequence>
3731 </xsd:extension>
3732 </xsd:complexContent>
3733 </xsd:complexType>
3734
3735 <xsd:complexType name="tLogicalPeopleGroup">
3736 <xsd:complexContent>
3737 <xsd:extension base="tExtensibleElements">
3738 <xsd:sequence>
3739 <xsd:element name="parameter" type="tParameter" minOccurs="0"
3740 maxOccurs="unbounded" />
3741 </xsd:sequence>
3742 <xsd:attribute name="name" type="xsd:NCName" use="required" />
3743 <xsd:attribute name="reference" type="xsd:NCName" use="optional" />
3744 </xsd:extension>
3745 </xsd:complexContent>
3746 </xsd:complexType>
3747
3748 <!-- generic human roles used in tasks and notifications -->
3749 <xsd:element name="genericHumanRole" type="tGenericHumanRoleAssignmentBase"
3750 abstract="true" block="" />
3751
3752 <xsd:element name="potentialOwners" type="tPotentialOwnerAssignment"
3753 substitutionGroup="genericHumanRole" />
3754 <xsd:element name="excludedOwners" type="tGenericHumanRoleAssignment"
3755 substitutionGroup="genericHumanRole" />
3756 <xsd:element name="taskInitiator" type="tGenericHumanRoleAssignment"
3757 substitutionGroup="genericHumanRole" />
3758 <xsd:element name="taskStakeholders" type="tGenericHumanRoleAssignment"
3759 substitutionGroup="genericHumanRole" />
3760 <xsd:element name="businessAdministrators"
3761 type="tGenericHumanRoleAssignment" substitutionGroup="genericHumanRole" />
3762 <xsd:element name="recipients" type="tGenericHumanRoleAssignment"
3763 substitutionGroup="genericHumanRole" />
3764
3765 <xsd:complexType name="tGenericHumanRoleAssignmentBase" block="">
3766 <xsd:complexContent>
3767 <xsd:extension base="tExtensibleElements" />
3768 </xsd:complexContent>
3769 </xsd:complexType>
3770
3771 <xsd:complexType name="tGenericHumanRoleAssignment">
3772 <xsd:complexContent>
3773 <xsd:extension base="tGenericHumanRoleAssignmentBase">
3774 <xsd:sequence>
3775 <xsd:element name="from" type="tFrom" />
3776 </xsd:sequence>
3777 </xsd:extension>
3778 </xsd:complexContent>
3779 </xsd:complexType>
3780
3781 <xsd:complexType name="tPotentialOwnerAssignment">
3782 <xsd:complexContent>
3783 <xsd:extension base="tGenericHumanRoleAssignmentBase">
3784 <xsd:choice>
3785 <xsd:element name="from" type="tFrom" />

```

```

3786     <xsd:element name="parallel" type="tParallel" />
3787     <xsd:element name="sequence" type="tSequence" />
3788   </xsd:choice>
3789 </xsd:extension>
3790 </xsd:complexContent>
3791 </xsd:complexType>
3792
3793 <!-- routing patterns -->
3794 <xsd:complexType name="tParallel">
3795   <xsd:complexContent>
3796     <xsd:extension base="tExtensibleElements">
3797       <xsd:sequence>
3798         <xsd:element name="completionBehavior" type="tCompletionBehavior"
3799 minOccurs="0" />
3800         <xsd:element name="from" type="tFrom" minOccurs="0"
3801 maxOccurs="unbounded" />
3802         <xsd:choice minOccurs="0" maxOccurs="unbounded">
3803           <xsd:element name="parallel" type="tParallel" />
3804           <xsd:element name="sequence" type="tSequence" />
3805         </xsd:choice>
3806       </xsd:sequence>
3807       <xsd:attribute name="type" type="tRoutingPatternType" />
3808     </xsd:extension>
3809   </xsd:complexContent>
3810 </xsd:complexType>
3811
3812 <xsd:complexType name="tSequence">
3813   <xsd:complexContent>
3814     <xsd:extension base="tExtensibleElements">
3815       <xsd:sequence>
3816         <xsd:element name="completionBehavior" type="tCompletionBehavior"
3817 />
3818         <xsd:element name="from" type="tFrom" minOccurs="0"
3819 maxOccurs="unbounded" />
3820         <xsd:choice minOccurs="0" maxOccurs="unbounded">
3821           <xsd:element name="parallel" type="tParallel" />
3822           <xsd:element name="sequence" type="tSequence" />
3823         </xsd:choice>
3824       </xsd:sequence>
3825       <xsd:attribute name="type" type="tRoutingPatternType" />
3826     </xsd:extension>
3827   </xsd:complexContent>
3828 </xsd:complexType>
3829
3830 <xsd:simpleType name="tRoutingPatternType">
3831   <xsd:restriction base="xsd:string">
3832     <xsd:enumeration value="all" />
3833     <xsd:enumeration value="single" />
3834   </xsd:restriction>
3835 </xsd:simpleType>
3836
3837 <!-- completion behavior -->
3838 <xsd:complexType name="tCompletionBehavior">
3839   <xsd:complexContent>
3840     <xsd:extension base="tExtensibleElements">
3841       <xsd:sequence>
3842         <xsd:element name="completion" type="tCompletion" minOccurs="0"
3843 maxOccurs="unbounded" />

```



```

3844         <xsd:element name="defaultCompletion" type="tDefaultCompletion"
3845 minOccurs="0" />
3846     </xsd:sequence>
3847     <xsd:attribute name="completionAction" type="tPattern" use="optional"
3848 default="automatic" />
3849 </xsd:extension>
3850 </xsd:complexContent>
3851 </xsd:complexType>
3852
3853 <xsd:complexType name="tCompletion">
3854 <xsd:complexContent>
3855 <xsd:extension base="tExtensibleElements">
3856 <xsd:sequence>
3857 <xsd:element name="condition" type="tBoolean-expr" />
3858 <xsd:element name="result" type="tResult" minOccurs="0" />
3859 </xsd:sequence>
3860 </xsd:extension>
3861 </xsd:complexContent>
3862 </xsd:complexType>
3863
3864 <xsd:complexType name="tDefaultCompletion">
3865 <xsd:complexContent>
3866 <xsd:extension base="tExtensibleElements">
3867 <xsd:sequence>
3868 <xsd:element name="result" type="tResult" />
3869 </xsd:sequence>
3870 </xsd:extension>
3871 </xsd:complexContent>
3872 </xsd:complexType>
3873
3874 <!-- result construction -->
3875 <xsd:complexType name="tResult">
3876 <xsd:complexContent>
3877 <xsd:extension base="tExtensibleElements">
3878 <xsd:choice maxOccurs="unbounded">
3879 <xsd:element name="aggregate" type="tAggregate" />
3880 <xsd:element name="copy" type="tCopy" />
3881 </xsd:choice>
3882 </xsd:extension>
3883 </xsd:complexContent>
3884 </xsd:complexType>
3885
3886 <xsd:complexType name="tAggregate">
3887 <xsd:complexContent>
3888 <xsd:extension base="tExtensibleElements">
3889 <xsd:attribute name="part" type="xsd:NCName" use="optional" />
3890 <xsd:attribute name="location" type="xsd:string" use="optional" />
3891 <xsd:attribute name="condition" type="xsd:string" />
3892 <xsd:attribute name="function" type="xsd:string" use="required" />
3893 </xsd:extension>
3894 </xsd:complexContent>
3895 </xsd:complexType>
3896
3897 <xsd:complexType name="tCopy">
3898 <xsd:complexContent>
3899 <xsd:extension base="tExtensibleElements">
3900 <xsd:sequence>
3901 <xsd:element name="from" type="tExpression" />

```

```

3902     <xsd:element name="to" type="tQuery" />
3903   </xsd:sequence>
3904 </xsd:extension>
3905 </xsd:complexContent>
3906 </xsd:complexType>
3907
3908 <!-- human tasks -->
3909 <xsd:element name="tasks" type="tTasks" />
3910 <xsd:complexType name="tTasks">
3911   <xsd:complexContent>
3912     <xsd:extension base="tExtensibleElements">
3913       <xsd:sequence>
3914         <xsd:element name="task" type="tTask" maxOccurs="unbounded" />
3915       </xsd:sequence>
3916     </xsd:extension>
3917   </xsd:complexContent>
3918 </xsd:complexType>
3919
3920 <xsd:complexType name="tTaskBase" abstract="true">
3921   <xsd:complexContent>
3922     <xsd:extension base="tExtensibleElements">
3923       <xsd:sequence>
3924         <xsd:element name="interface" type="tTaskInterface" minOccurs="0"
3925 />
3926         <xsd:element name="messageSchema" type="tMessageSchema"
3927 minOccurs="0" />
3928         <xsd:element name="priority" type="tPriority-expr" minOccurs="0" />
3929         <xsd:element name="peopleAssignments" type="tPeopleAssignments"
3930 minOccurs="0" />
3931         <xsd:element name="completionBehavior" type="tCompletionBehavior"
3932 minOccurs="0" />
3933         <xsd:element name="delegation" type="tDelegation" minOccurs="0" />
3934         <xsd:element name="presentationElements"
3935 type="tPresentationElements" minOccurs="0" />
3936         <xsd:element name="outcome" type="tQuery" minOccurs="0" />
3937         <xsd:element name="searchBy" type="tExpression" minOccurs="0" />
3938         <xsd:element name="renderings" type="tRenderings" minOccurs="0" />
3939         <xsd:element name="deadlines" type="tDeadlines" minOccurs="0" />
3940         <xsd:element name="composition" type="tComposition" minOccurs="0"
3941 />
3942       </xsd:sequence>
3943       <xsd:attribute name="name" type="xsd:NCName" use="required" />
3944       <xsd:attribute name="actualOwnerRequired" type="tBoolean"
3945 use="optional" default="yes" />
3946     </xsd:extension>
3947   </xsd:complexContent>
3948 </xsd:complexType>
3949
3950 <xsd:element name="task" type="tTask" />
3951 <xsd:complexType name="tTask">
3952   <xsd:complexContent>
3953     <xsd:restriction base="tTaskBase">
3954       <xsd:sequence>
3955         <xsd:element name="documentation" type="tDocumentation"
3956 minOccurs="0" maxOccurs="unbounded" />
3957         <xsd:any namespace="##other" processContents="lax" minOccurs="0"
3958 maxOccurs="unbounded" />
3959         <xsd:element name="interface" type="tTaskInterface" />

```

```

3960     <xsd:element name="messageSchema" type="tMessageSchema"
3961 minOccurs="0" maxOccurs="0" />
3962     <xsd:element name="priority" type="tPriority-expr" minOccurs="0" />
3963     <xsd:element name="peopleAssignments" type="tPeopleAssignments"
3964 minOccurs="0" />
3965     <xsd:element name="completionBehavior" type="tCompletionBehavior"
3966 minOccurs="0" />
3967     <xsd:element name="delegation" type="tDelegation" minOccurs="0" />
3968     <xsd:element name="presentationElements"
3969 type="tPresentationElements" minOccurs="0" />
3970     <xsd:element name="outcome" type="tQuery" minOccurs="0" />
3971     <xsd:element name="searchBy" type="tExpression" minOccurs="0" />
3972     <xsd:element name="renderings" type="tRenderings" minOccurs="0" />
3973     <xsd:element name="deadlines" type="tDeadlines" minOccurs="0" />
3974     <xsd:element name="composition" type="tComposition" minOccurs="0"
3975 />
3976     </xsd:sequence>
3977     <xsd:attribute name="name" type="xsd:NCName" use="required" />
3978     <xsd:attribute name="actualOwnerRequired" type="tBoolean"
3979 use="optional" default="yes" />
3980     <xsd:anyAttribute namespace="##other" processContents="lax" />
3981     </xsd:restriction>
3982     </xsd:complexContent>
3983     </xsd:complexType>
3984
3985     <xsd:complexType name="tTaskInterface">
3986     <xsd:complexContent>
3987     <xsd:extension base="tExtensibleElements">
3988     <xsd:attribute name="portType" type="xsd:QName" use="required" />
3989     <xsd:attribute name="operation" type="xsd:NCName" use="required" />
3990     <xsd:attribute name="responsePortType" type="xsd:QName"
3991 use="optional" />
3992     <xsd:attribute name="responseOperation" type="xsd:NCName"
3993 use="optional" />
3994     </xsd:extension>
3995     </xsd:complexContent>
3996     </xsd:complexType>
3997
3998     <!-- presentation elements -->
3999     <xsd:complexType name="tPresentationElements">
4000     <xsd:complexContent>
4001     <xsd:extension base="tExtensibleElements">
4002     <xsd:sequence>
4003     <xsd:element name="name" type="tText" minOccurs="0"
4004 maxOccurs="unbounded" />
4005     <xsd:element name="presentationParameters"
4006 type="tPresentationParameters" minOccurs="0" />
4007     <xsd:element name="subject" type="tText" minOccurs="0"
4008 maxOccurs="unbounded" />
4009     <xsd:element name="description" type="tDescription" minOccurs="0"
4010 maxOccurs="unbounded" />
4011     </xsd:sequence>
4012     </xsd:extension>
4013     </xsd:complexContent>
4014     </xsd:complexType>
4015
4016     <xsd:complexType name="tPresentationParameters">
4017     <xsd:complexContent>

```

```

4018     <xsd:extension base="tExtensibleElements">
4019         <xsd:sequence>
4020             <xsd:element name="presentationParameter"
4021 type="tPresentationParameter" maxOccurs="unbounded" />
4022         </xsd:sequence>
4023         <xsd:attribute name="expressionLanguage" type="xsd:anyURI" />
4024     </xsd:extension>
4025 </xsd:complexContent>
4026 </xsd:complexType>
4027
4028 <xsd:complexType name="tPresentationParameter">
4029 <xsd:complexContent>
4030 <xsd:extension base="tParameter" />
4031 </xsd:complexContent>
4032 </xsd:complexType>
4033
4034 <!-- elements for rendering tasks -->
4035 <xsd:complexType name="tRenderings">
4036 <xsd:complexContent>
4037 <xsd:extension base="tExtensibleElements">
4038 <xsd:sequence>
4039 <xsd:element name="rendering" type="tRendering"
4040 maxOccurs="unbounded" />
4041 </xsd:sequence>
4042 </xsd:extension>
4043 </xsd:complexContent>
4044 </xsd:complexType>
4045
4046 <xsd:complexType name="tRendering">
4047 <xsd:complexContent>
4048 <xsd:extension base="tExtensibleElements">
4049 <xsd:attribute name="type" type="xsd:QName" use="required" />
4050 </xsd:extension>
4051 </xsd:complexContent>
4052 </xsd:complexType>
4053
4054 <!-- elements for people assignment -->
4055 <xsd:element name="peopleAssignments" type="tPeopleAssignments" />
4056 <xsd:complexType name="tPeopleAssignments">
4057 <xsd:complexContent>
4058 <xsd:extension base="tExtensibleElements">
4059 <xsd:sequence>
4060 <xsd:element ref="genericHumanRole" minOccurs="0"
4061 maxOccurs="unbounded" />
4062 </xsd:sequence>
4063 </xsd:extension>
4064 </xsd:complexContent>
4065 </xsd:complexType>
4066
4067 <!-- elements for handling timeouts and escalation -->
4068 <xsd:complexType name="tDeadlines">
4069 <xsd:complexContent>
4070 <xsd:extension base="tExtensibleElements">
4071 <xsd:sequence>
4072 <xsd:element name="startDeadline" type="tDeadline" minOccurs="0"
4073 maxOccurs="unbounded" />
4074 <xsd:element name="completionDeadline" type="tDeadline"
4075 minOccurs="0" maxOccurs="unbounded" />

```

```

4076     </xsd:sequence>
4077     </xsd:extension>
4078     </xsd:complexContent>
4079 </xsd:complexType>
4080
4081 <xsd:complexType name="tDeadline">
4082     <xsd:complexContent>
4083         <xsd:extension base="tExtensibleElements">
4084             <xsd:sequence>
4085                 <xsd:choice>
4086                     <xsd:element name="for" type="tDuration-expr" />
4087                     <xsd:element name="until" type="tDeadline-expr" />
4088                 </xsd:choice>
4089                 <xsd:element name="escalation" type="tEscalation" minOccurs="0"
4090 minOccurs="unbounded" />
4091             </xsd:sequence>
4092             <xsd:attribute name="name" type="xsd:NCName" use="required" />
4093         </xsd:extension>
4094     </xsd:complexContent>
4095 </xsd:complexType>
4096
4097 <xsd:complexType name="tEscalation">
4098     <xsd:complexContent>
4099         <xsd:extension base="tExtensibleElements">
4100             <xsd:sequence>
4101                 <xsd:element name="condition" type="tBoolean-expr" minOccurs="0" />
4102                 <xsd:element name="toParts" type="tToParts" minOccurs="0" />
4103             <xsd:choice>
4104                 <xsd:element name="notification" type="tNotification" />
4105                 <xsd:element name="localNotification" type="tLocalNotification"
4106 />
4107                 <xsd:element name="reassignment" type="tReassignment" />
4108             </xsd:choice>
4109         </xsd:sequence>
4110         <xsd:attribute name="name" type="xsd:NCName" use="required" />
4111     </xsd:extension>
4112 </xsd:complexContent>
4113 </xsd:complexType>
4114
4115 <xsd:complexType name="tLocalNotification">
4116     <xsd:complexContent>
4117         <xsd:extension base="tExtensibleElements">
4118             <xsd:choice>
4119                 <xsd:sequence>
4120                     <xsd:element name="priority" type="tPriority-expr" minOccurs="0"
4121 />
4122                     <xsd:element name="peopleAssignments" type="tPeopleAssignments"
4123 minOccurs="0" />
4124                 </xsd:sequence>
4125             </xsd:choice>
4126             <xsd:attribute name="reference" type="xsd:QName" use="required" />
4127         </xsd:extension>
4128     </xsd:complexContent>
4129 </xsd:complexType>
4130
4131 <xsd:complexType name="tReassignment">
4132     <xsd:complexContent>
4133         <xsd:extension base="tExtensibleElements">

```

```

4134     <xsd:sequence>
4135         <xsd:element ref="potentialOwners" />
4136     </xsd:sequence>
4137 </xsd:extension>
4138 </xsd:complexContent>
4139 </xsd:complexType>
4140
4141 <xsd:complexType name="tToParts">
4142     <xsd:complexContent>
4143         <xsd:extension base="tExtensibleElements">
4144             <xsd:sequence>
4145                 <xsd:element name="toPart" type="tToPart" maxOccurs="unbounded" />
4146             </xsd:sequence>
4147         </xsd:extension>
4148     </xsd:complexContent>
4149 </xsd:complexType>
4150
4151 <xsd:complexType name="tToPart" mixed="true">
4152     <xsd:complexContent>
4153         <xsd:extension base="tExtensibleMixedContentElements">
4154             <xsd:attribute name="name" type="xsd:NCName" use="required" />
4155             <xsd:attribute name="expressionLanguage" type="xsd:anyURI" />
4156         </xsd:extension>
4157     </xsd:complexContent>
4158 </xsd:complexType>
4159
4160 <!-- task delegation -->
4161 <xsd:complexType name="tDelegation">
4162     <xsd:complexContent>
4163         <xsd:extension base="tExtensibleElements">
4164             <xsd:sequence>
4165                 <xsd:element name="from" type="tFrom" minOccurs="0" />
4166             </xsd:sequence>
4167             <xsd:attribute name="potentialDelegates" type="tPotentialDelegates"
4168 use="required" />
4169         </xsd:extension>
4170     </xsd:complexContent>
4171 </xsd:complexType>
4172
4173 <xsd:simpleType name="tPotentialDelegates">
4174     <xsd:restriction base="xsd:string">
4175         <xsd:enumeration value="anybody" />
4176         <xsd:enumeration value="nobody" />
4177         <xsd:enumeration value="potentialOwners" />
4178         <xsd:enumeration value="other" />
4179     </xsd:restriction>
4180 </xsd:simpleType>
4181
4182 <!-- composite tasks -->
4183 <xsd:complexType name="tComposition">
4184     <xsd:complexContent>
4185         <xsd:extension base="tExtensibleElements">
4186             <xsd:sequence>
4187                 <xsd:element name="subtask" type="tSubtask" maxOccurs="unbounded"
4188 />
4189             </xsd:sequence>
4190             <xsd:attribute name="type" type="tCompositionType" use="optional"
4191 default="sequential" />

```

```

4192         <xsd:attribute name="instantiationPattern" type="tPattern"
4193 use="optional" default="manual" />
4194     </xsd:extension>
4195 </xsd:complexContent>
4196 </xsd:complexType>
4197
4198 <xsd:simpleType name="tCompositionType">
4199     <xsd:restriction base="xsd:string">
4200         <xsd:enumeration value="sequential" />
4201         <xsd:enumeration value="parallel" />
4202     </xsd:restriction>
4203 </xsd:simpleType>
4204
4205 <xsd:simpleType name="tPattern">
4206     <xsd:restriction base="xsd:string">
4207         <xsd:enumeration value="manual" />
4208         <xsd:enumeration value="automatic" />
4209     </xsd:restriction>
4210 </xsd:simpleType>
4211
4212 <xsd:complexType name="tSubtask">
4213     <xsd:complexContent>
4214         <xsd:extension base="tExtensibleElements">
4215             <xsd:choice>
4216                 <xsd:element name="task" type="tTask"/>
4217                 <xsd:element name="localTask" type="tLocalTask" />
4218             </xsd:choice>
4219             <xsd:attribute name="name" type="xsd:NCName" use="required" />
4220         </xsd:extension>
4221     </xsd:complexContent>
4222 </xsd:complexType>
4223
4224 <xsd:complexType name="tLocalTask">
4225     <xsd:complexContent>
4226         <xsd:extension base="tExtensibleElements">
4227             <xsd:sequence>
4228                 <xsd:element name="priority" type="tPriority-expr" minOccurs="0" />
4229                 <xsd:element name="peopleAssignments" type="tPeopleAssignments"
4230 minOccurs="0" />
4231             </xsd:sequence>
4232             <xsd:attribute name="reference" type="xsd:QName" use="required" />
4233         </xsd:extension>
4234     </xsd:complexContent>
4235 </xsd:complexType>
4236
4237 <!-- lean tasks -->
4238 <xsd:element name="leanTask" type="tLeanTask"/>
4239 <xsd:complexType name="tLeanTask">
4240     <xsd:complexContent>
4241         <xsd:restriction base="tTaskBase">
4242             <xsd:sequence>
4243                 <xsd:element name="documentation" type="tDocumentation"
4244 minOccurs="0" maxOccurs="unbounded" />
4245                 <xsd:any namespace="##other" processContents="lax" minOccurs="0"
4246 maxOccurs="unbounded" />
4247                 <xsd:element name="interface" type="tTaskInterface" minOccurs="0"
4248 maxOccurs="0" />
4249                 <xsd:element name="messageSchema" type="tMessageSchema" />

```



```

4250     <xsd:element name="priority" type="tPriority-expr" minOccurs="0" />
4251     <xsd:element name="peopleAssignments" type="tPeopleAssignments"
4252 minOccurs="0" />
4253     <xsd:element name="delegation" type="tDelegation" minOccurs="0" />
4254     <xsd:element name="presentationElements"
4255 type="tPresentationElements" minOccurs="0" />
4256     <xsd:element name="outcome" type="tQuery" minOccurs="0" />
4257     <xsd:element name="searchBy" type="tExpression" minOccurs="0" />
4258     <xsd:element name="renderings" type="tRenderings" minOccurs="0" />
4259     <xsd:element name="deadlines" type="tDeadlines" minOccurs="0" />
4260     <xsd:element name="composition" type="tComposition" minOccurs="0"
4261 maxOccurs="0" />
4262     </xsd:sequence>
4263     <xsd:attribute name="name" type="xsd:NCName" use="required" />
4264     <xsd:attribute name="actualOwnerRequired" type="tBoolean"
4265 use="optional" default="yes" />
4266     <xsd:anyAttribute namespace="##other" processContents="lax" />
4267     </xsd:restriction>
4268     </xsd:complexContent>
4269     </xsd:complexType>
4270
4271     <xsd:complexType name="tMessageSchema">
4272     <xsd:complexContent>
4273     <xsd:extension base="tExtensibleElements">
4274     <xsd:sequence>
4275     <xsd:element name="messageField" type="tMessageField"
4276 minOccurs="0" maxOccurs="unbounded" />
4277     </xsd:sequence>
4278     </xsd:extension>
4279     </xsd:complexContent>
4280     </xsd:complexType>
4281
4282     <xsd:complexType name="tMessageField">
4283     <xsd:complexContent>
4284     <xsd:extension base="tExtensibleElements">
4285     <xsd:sequence>
4286     <xsd:element name="messageDisplay" type="tMessageDisplay"
4287 maxOccurs="unbounded" />
4288     <xsd:element name="messageChoice" type="tMessageChoice"
4289 minOccurs="0" maxOccurs="unbounded" />
4290     </xsd:sequence>
4291     <xsd:attribute name="name" type="xsd:NCName" />
4292     <xsd:attribute name="type" type="xsd:QName" />
4293     </xsd:extension>
4294     </xsd:complexContent>
4295     </xsd:complexType>
4296
4297     <xsd:complexType name="tMessageChoice">
4298     <xsd:complexContent>
4299     <xsd:extension base="tExtensibleElements">
4300     <xsd:sequence>
4301     <xsd:element name="messageDisplay" type="tMessageDisplay"
4302 maxOccurs="unbounded" />
4303     </xsd:sequence>
4304     </xsd:extension>
4305     </xsd:complexContent>
4306     </xsd:complexType>
4307

```



```

4308 <xsd:complexType name="tMessageDisplay">
4309   <xsd:complexContent>
4310     <xsd:extension base="tExtensibleElements">
4311       <xsd:attribute ref="xml:lang" />
4312     </xsd:extension>
4313   </xsd:complexContent>
4314 </xsd:complexType>
4315
4316 <!-- notifications -->
4317 <xsd:element name="notifications" type="tNotifications" />
4318 <xsd:complexType name="tNotifications">
4319   <xsd:complexContent>
4320     <xsd:extension base="tExtensibleElements">
4321       <xsd:sequence>
4322         <xsd:element name="notification" type="tNotification"
4323 maxOccurs="unbounded" />
4324       </xsd:sequence>
4325     </xsd:extension>
4326   </xsd:complexContent>
4327 </xsd:complexType>
4328
4329 <xsd:element name="notification" type="tNotification" />
4330 <xsd:complexType name="tNotification">
4331   <xsd:complexContent>
4332     <xsd:extension base="tExtensibleElements">
4333       <xsd:sequence>
4334         <xsd:element name="interface" type="tNotificationInterface" />
4335         <xsd:element name="priority" type="tPriority-expr" minOccurs="0" />
4336         <xsd:element name="peopleAssignments" type="tPeopleAssignments" />
4337         <xsd:element name="presentationElements"
4338 type="tPresentationElements" />
4339         <xsd:element name="renderings" type="tRenderings" minOccurs="0" />
4340       </xsd:sequence>
4341       <xsd:attribute name="name" type="xsd:NCName" use="required" />
4342     </xsd:extension>
4343   </xsd:complexContent>
4344 </xsd:complexType>
4345
4346 <xsd:complexType name="tNotificationInterface">
4347   <xsd:complexContent>
4348     <xsd:extension base="tExtensibleElements">
4349       <xsd:attribute name="portType" type="xsd:QName" use="required" />
4350       <xsd:attribute name="operation" type="xsd:NCName" use="required" />
4351     </xsd:extension>
4352   </xsd:complexContent>
4353 </xsd:complexType>
4354
4355 <!-- miscellaneous helper types -->
4356 <xsd:complexType name="tText" mixed="true">
4357   <xsd:complexContent>
4358     <xsd:extension base="tExtensibleMixedContentElements">
4359       <xsd:attribute ref="xml:lang" />
4360     </xsd:extension>
4361   </xsd:complexContent>
4362 </xsd:complexType>
4363
4364 <xsd:complexType name="tDescription" mixed="true">
4365   <xsd:complexContent>

```

```

4366     <xsd:extension base="tExtensibleMixedContentElements">
4367         <xsd:attribute ref="xml:lang" />
4368         <xsd:attribute name="contentType" type="xsd:string" />
4369     </xsd:extension>
4370 </xsd:complexContent>
4371 </xsd:complexType>
4372
4373 <xsd:complexType name="tFrom" mixed="true">
4374     <xsd:complexContent>
4375         <xsd:extension base="tExtensibleMixedContentElements">
4376             <xsd:sequence>
4377                 <xsd:choice>
4378                     <xsd:element name="argument" type="tArgument" minOccurs="0"
4379 maxOccurs="unbounded" />
4380                     <xsd:element name="literal" type="tLiteral" minOccurs="0" />
4381                 </xsd:choice>
4382             </xsd:sequence>
4383             <xsd:attribute name="expressionLanguage" type="xsd:anyURI" />
4384             <xsd:attribute name="logicalPeopleGroup" type="xsd:NCName" />
4385         </xsd:extension>
4386     </xsd:complexContent>
4387 </xsd:complexType>
4388
4389 <xsd:complexType name="tArgument">
4390     <xsd:complexContent>
4391         <xsd:extension base="tExtensibleMixedContentElements">
4392             <xsd:attribute name="name" type="xsd:NCName" />
4393             <xsd:attribute name="expressionLanguage" type="xsd:anyURI" />
4394         </xsd:extension>
4395     </xsd:complexContent>
4396 </xsd:complexType>
4397
4398 <xsd:complexType name="tParameter" mixed="true">
4399     <xsd:complexContent>
4400         <xsd:extension base="tExtensibleMixedContentElements">
4401             <xsd:attribute name="name" type="xsd:NCName" use="required" />
4402             <xsd:attribute name="type" type="xsd:QName" use="required" />
4403         </xsd:extension>
4404     </xsd:complexContent>
4405 </xsd:complexType>
4406
4407 <xsd:complexType name="tLiteral" mixed="true">
4408     <xsd:sequence>
4409         <xsd:any namespace="##any" processContents="lax" />
4410     </xsd:sequence>
4411     <xsd:anyAttribute namespace="##other" processContents="lax" />
4412 </xsd:complexType>
4413
4414 <xsd:complexType name="tQuery" mixed="true">
4415     <xsd:complexContent>
4416         <xsd:extension base="tExtensibleMixedContentElements">
4417             <xsd:attribute name="part" />
4418             <xsd:attribute name="queryLanguage" type="xsd:anyURI" />
4419         </xsd:extension>
4420     </xsd:complexContent>
4421 </xsd:complexType>
4422
4423 <xsd:complexType name="tExpression" mixed="true">

```

```
4424     <xsd:complexContent>
4425         <xsd:extension base="tExtensibleMixedContentElements">
4426             <xsd:attribute name="expressionLanguage" type="xsd:anyURI" />
4427         </xsd:extension>
4428     </xsd:complexContent>
4429 </xsd:complexType>
4430
4431 <xsd:element name="priority" type="tPriority-expr" />
4432 <xsd:complexType name="tPriority-expr" mixed="true">
4433     <xsd:complexContent mixed="true">
4434         <xsd:extension base="tExpression" />
4435     </xsd:complexContent>
4436 </xsd:complexType>
4437
4438 <xsd:complexType name="tBoolean-expr" mixed="true">
4439     <xsd:complexContent mixed="true">
4440         <xsd:extension base="tExpression" />
4441     </xsd:complexContent>
4442 </xsd:complexType>
4443
4444 <xsd:complexType name="tDuration-expr" mixed="true">
4445     <xsd:complexContent mixed="true">
4446         <xsd:extension base="tExpression" />
4447     </xsd:complexContent>
4448 </xsd:complexType>
4449
4450 <xsd:complexType name="tDeadline-expr" mixed="true">
4451     <xsd:complexContent mixed="true">
4452         <xsd:extension base="tExpression" />
4453     </xsd:complexContent>
4454 </xsd:complexType>
4455
4456 <xsd:simpleType name="tBoolean">
4457     <xsd:restriction base="xsd:string">
4458         <xsd:enumeration value="yes" />
4459         <xsd:enumeration value="no" />
4460     </xsd:restriction>
4461 </xsd:simpleType>
4462
4463 </xsd:schema>
```

4464

C. WS-HumanTask Data Types Schema

```

4465 <?xml version="1.0" encoding="UTF-8"?>
4466 <!--
4467 Copyright (c) OASIS Open 2009. All Rights Reserved.
4468 -->
4469 <xsd:schema
4470   targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
4471 humantask/types/200803"
4472   xmlns="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/types/200803"
4473   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
4474   elementFormDefault="qualified"
4475   blockDefault="#all">
4476
4477   <xsd:annotation>
4478     <xsd:documentation>
4479       XML Schema for WS-HumanTask 1.1 - WS-HumanTask Data Type Definitions
4480     </xsd:documentation>
4481   </xsd:annotation>
4482
4483   <!-- other namespaces -->
4484   <xsd:import namespace="http://www.w3.org/XML/1998/namespace"
4485 schemaLocation="http://www.w3.org/2001/xml.xsd" />
4486
4487   <!-- data types for attachment operations -->
4488   <xsd:element name="attachmentInfo" type="tAttachmentInfo" />
4489   <xsd:complexType name="tAttachmentInfo">
4490     <xsd:sequence>
4491       <xsd:element name="identifier" type="xsd:anyURI" />
4492       <xsd:element name="name" type="xsd:string" />
4493       <xsd:element name="accessType" type="xsd:string" />
4494       <xsd:element name="contentType" type="xsd:string" />
4495       <xsd:element name="contentCategory" type="xsd:anyURI" />
4496       <xsd:element name="attachedTime" type="xsd:dateTime" />
4497       <xsd:element name="attachedBy" type="tUser" />
4498       <xsd:any namespace="##other" processContents="lax" minOccurs="0"
4499 maxOccurs="unbounded" />
4500     </xsd:sequence>
4501   </xsd:complexType>
4502   <xsd:element name="attachment" type="tAttachment" />
4503   <xsd:complexType name="tAttachment">
4504     <xsd:sequence>
4505       <xsd:element ref="attachmentInfo" />
4506       <xsd:element name="value" type="xsd:anyType" />
4507     </xsd:sequence>
4508   </xsd:complexType>
4509
4510   <!-- data types for comments -->
4511   <xsd:element name="comment" type="tComment" />
4512   <xsd:complexType name="tComment">
4513     <xsd:sequence>
4514       <xsd:element name="addedTime" type="xsd:dateTime" />
4515       <xsd:element name="addedBy" type="tUser" />
4516       <xsd:element name="text" type="xsd:string" />

```

```

4517     <xsd:any namespace="##other" processContents="lax" minOccurs="0"
4518 maxOccurs="unbounded" />
4519     </xsd:sequence>
4520 </xsd:complexType>
4521
4522 <!-- data types for simple query operations -->
4523 <xsd:element name="taskAbstract" type="tTaskAbstract" />
4524 <xsd:complexType name="tTaskAbstract">
4525     <xsd:sequence>
4526         <xsd:element name="id" type="xsd:string" />
4527         <xsd:element name="taskType" type="xsd:string" />
4528         <xsd:element name="name" type="xsd:QName" />
4529         <xsd:element name="status" type="tStatus" />
4530         <xsd:element name="priority" type="tPriority" minOccurs="0" />
4531         <xsd:element name="createdTime" type="xsd:dateTime" />
4532         <xsd:element name="activationTime" type="xsd:dateTime" minOccurs="0" />
4533         <xsd:element name="expirationTime" type="xsd:dateTime" minOccurs="0" />
4534         <xsd:element name="isSkipable" type="xsd:boolean" minOccurs="0" />
4535         <xsd:element name="hasPotentialOwners" type="xsd:boolean"
4536 minOccurs="0" />
4537         <xsd:element name="startByTimeExists" type="xsd:boolean"
4538 minOccurs="0" />
4539         <xsd:element name="completeByTimeExists" type="xsd:boolean"
4540 minOccurs="0" />
4541         <xsd:element name="presentationName" type="tPresentationName"
4542 minOccurs="0" />
4543         <xsd:element name="presentationSubject" type="tPresentationSubject"
4544 minOccurs="0" />
4545         <xsd:element name="renderingMethodExists" type="xsd:boolean" />
4546         <xsd:element name="hasOutput" type="xsd:boolean" minOccurs="0" />
4547         <xsd:element name="hasFault" type="xsd:boolean" minOccurs="0" />
4548         <xsd:element name="hasAttachments" type="xsd:boolean" minOccurs="0" />
4549         <xsd:element name="hasComments" type="xsd:boolean" minOccurs="0" />
4550         <xsd:element name="escalated" type="xsd:boolean" minOccurs="0" />
4551         <xsd:element name="outcome" type="xsd:string" minOccurs="0" />
4552         <xsd:element name="parentTaskId" type="xsd:string" minOccurs="0" />
4553         <xsd:element name="hasSubTasks" type="xsd:boolean" minOccurs="0" />
4554     <xsd:any namespace="##other" processContents="lax" minOccurs="0"
4555 maxOccurs="unbounded" />
4556     </xsd:sequence>
4557 </xsd:complexType>
4558 <xsd:element name="taskDetails" type="tTaskDetails" />
4559 <xsd:complexType name="tTaskDetails">
4560     <xsd:sequence>
4561         <xsd:element name="id" type="xsd:string" />
4562         <xsd:element name="taskType" type="xsd:string" />
4563         <xsd:element name="name" type="xsd:QName" />
4564         <xsd:element name="status" type="tStatus" />
4565         <xsd:element name="priority" type="tPriority" minOccurs="0" />
4566         <xsd:element name="taskInitiator" type="tUser" minOccurs="0" />
4567         <xsd:element name="taskStakeholders" type="tOrganizationalEntity"
4568 minOccurs="0" />
4569         <xsd:element name="potentialOwners" type="tOrganizationalEntity"
4570 minOccurs="0" />
4571         <xsd:element name="businessAdministrators" type="tOrganizationalEntity"
4572 minOccurs="0" />
4573         <xsd:element name="actualOwner" type="tUser" minOccurs="0" />

```

```

4574     <xsd:element name="notificationRecipients" type="tOrganizationalEntity"
4575 minOccurs="0"/>
4576     <xsd:element name="createdTime" type="xsd:dateTime"/>
4577     <xsd:element name="createdBy" type="xsd:string" minOccurs="0"/>
4578     <xsd:element name="lastModifiedTime" type="xsd:dateTime"/>
4579     <xsd:element name="lastModifiedBy" type="xsd:string" minOccurs="0"/>
4580     <xsd:element name="activationTime" type="xsd:dateTime" minOccurs="0"/>
4581     <xsd:element name="expirationTime" type="xsd:dateTime" minOccurs="0"/>
4582     <xsd:element name="isSkipable" type="xsd:boolean" minOccurs="0"/>
4583     <xsd:element name="hasPotentialOwners" type="xsd:boolean"
4584 minOccurs="0"/>
4585     <xsd:element name="startByTimeExists" type="xsd:boolean"
4586 minOccurs="0"/>
4587     <xsd:element name="completeByTimeExists" type="xsd:boolean"
4588 minOccurs="0"/>
4589     <xsd:element name="presentationName" type="tPresentationName"
4590 minOccurs="0"/>
4591     <xsd:element name="presentationSubject" type="tPresentationSubject"
4592 minOccurs="0"/>
4593     <xsd:element name="renderingMethodExists" type="xsd:boolean"/>
4594     <xsd:element name="hasOutput" type="xsd:boolean" minOccurs="0"/>
4595     <xsd:element name="hasFault" type="xsd:boolean" minOccurs="0"/>
4596     <xsd:element name="hasAttachments" type="xsd:boolean" minOccurs="0"/>
4597     <xsd:element name="hasComments" type="xsd:boolean" minOccurs="0"/>
4598     <xsd:element name="escalated" type="xsd:boolean" minOccurs="0"/>
4599     <xsd:element name="searchBy" type="xsd:string" minOccurs="0"/>
4600     <xsd:element name="outcome" type="xsd:string" minOccurs="0"/>
4601     <xsd:element name="parentTaskId" type="xsd:string" minOccurs="0"/>
4602     <xsd:element name="hasSubTasks" type="xsd:boolean" minOccurs="0"/>
4603     <xsd:any namespace="##other" processContents="lax" minOccurs="0"
4604 maxOccurs="unbounded"/>
4605   </xsd:sequence>
4606 </xsd:complexType>
4607 <xsd:simpleType name="tPresentationName">
4608   <xsd:annotation>
4609     <xsd:documentation>length-restricted string</xsd:documentation>
4610   </xsd:annotation>
4611   <xsd:restriction base="xsd:string">
4612     <xsd:maxLength value="64"/>
4613     <xsd:whiteSpace value="preserve"/>
4614   </xsd:restriction>
4615 </xsd:simpleType>
4616 <xsd:simpleType name="tPresentationSubject">
4617   <xsd:annotation>
4618     <xsd:documentation>length-restricted string</xsd:documentation>
4619   </xsd:annotation>
4620   <xsd:restriction base="xsd:string">
4621     <xsd:maxLength value="254"/>
4622     <xsd:whiteSpace value="preserve"/>
4623   </xsd:restriction>
4624 </xsd:simpleType>
4625 <xsd:simpleType name="tStatus">
4626   <xsd:restriction base="xsd:string"/>
4627 </xsd:simpleType>
4628 <xsd:simpleType name="tPredefinedStatus">
4629   <xsd:annotation>
4630     <xsd:documentation>for documentation only</xsd:documentation>
4631   </xsd:annotation>

```



```

4632 <xsd:restriction base="xsd:string">
4633   <xsd:enumeration value="CREATED" />
4634   <xsd:enumeration value="READY" />
4635   <xsd:enumeration value="RESERVED" />
4636   <xsd:enumeration value="IN_PROGRESS" />
4637   <xsd:enumeration value="SUSPENDED" />
4638   <xsd:enumeration value="COMPLETED" />
4639   <xsd:enumeration value="FAILED" />
4640   <xsd:enumeration value="ERROR" />
4641   <xsd:enumeration value="EXITED" />
4642   <xsd:enumeration value="OBSOLETE" />
4643 </xsd:restriction>
4644 </xsd:simpleType>
4645 <xsd:simpleType name="tPriority">
4646   <xsd:restriction base="xsd:integer">
4647     <xsd:minInclusive value="0" />
4648     <xsd:maxInclusive value="10" />
4649   </xsd:restriction>
4650 </xsd:simpleType>
4651 <xsd:complexType name="tTime">
4652   <xsd:choice>
4653     <xsd:element name="timePeriod" type="xsd:duration" />
4654     <xsd:element name="pointOfTime" type="xsd:dateTime" />
4655   </xsd:choice>
4656 </xsd:complexType>
4657
4658 <!-- task operations -->
4659 <xsd:complexType name="tTaskOperations">
4660   <xsd:choice maxOccurs="unbounded">
4661     <xsd:element name="claim" type="tTaskOperation" />
4662     <xsd:element name="start" type="tTaskOperation" />
4663     <xsd:element name="stop" type="tTaskOperation" />
4664     <xsd:element name="release" type="tTaskOperation" />
4665     <xsd:element name="suspend" type="tTaskOperation" />
4666     <xsd:element name="suspendUntil" type="tTaskOperation" />
4667     <xsd:element name="resume" type="tTaskOperation" />
4668     <xsd:element name="complete" type="tTaskOperation" />
4669     <xsd:element name="remove" type="tTaskOperation" />
4670     <xsd:element name="fail" type="tTaskOperation" />
4671     <xsd:element name="setPriority" type="tTaskOperation" />
4672     <xsd:element name="addAttachment" type="tTaskOperation" />
4673     <xsd:element name="getAttachmentInfos" type="tTaskOperation" />
4674     <xsd:element name="getAttachment" type="tTaskOperation" />
4675     <xsd:element name="deleteAttachment" type="tTaskOperation" />
4676     <xsd:element name="addComment" type="tTaskOperation" />
4677     <xsd:element name="updateComment" type="tTaskOperation" />
4678     <xsd:element name="deleteComment" type="tTaskOperation" />
4679     <xsd:element name="getComments" type="tTaskOperation" />
4680     <xsd:element name="skip" type="tTaskOperation" />
4681     <xsd:element name="forward" type="tTaskOperation" />
4682     <xsd:element name="delegate" type="tTaskOperation" />
4683     <xsd:element name="getRendering" type="tTaskOperation" />
4684     <xsd:element name="getRenderingTypes" type="tTaskOperation" />
4685     <xsd:element name="getTaskDetails" type="tTaskOperation" />
4686     <xsd:element name="getTaskDescription" type="tTaskOperation" />
4687     <xsd:element name="setOutput" type="tTaskOperation" />
4688     <xsd:element name="deleteOutput" type="tTaskOperation" />
4689     <xsd:element name="setFault" type="tTaskOperation" />

```

```

4690     <xsd:element name="deleteFault" type="tTaskOperation" />
4691     <xsd:element name="getInput" type="tTaskOperation" />
4692     <xsd:element name="getOutput" type="tTaskOperation" />
4693     <xsd:element name="getFault" type="tTaskOperation" />
4694     <xsd:element name="getOutcome" type="tTaskOperation" />
4695     <xsd:element name="getTaskHistory" type="tTaskOperation" />
4696     <xsd:element name="getTaskInstanceData" type="tTaskOperation" />
4697     <xsd:element name="getSubtasks" type="tTaskOperation" />
4698     <xsd:element name="getSubtaskIdentifiers" type="tTaskOperation" />
4699     <xsd:element name="hasSubtasks" type="tTaskOperation" />
4700     <xsd:element name="getParentTask" type="tTaskOperation" />
4701     <xsd:element name="getParentTaskIdentifier" type="tTaskOperation" />
4702     <xsd:element name="isSubtask" type="tTaskOperation" />
4703     <xsd:element name="instantiateSubtask" type="tTaskOperation" />
4704     <xsd:element name="setTaskStartDeadlineExpression"
4705 type="tTaskOperation" />
4706     <xsd:element name="setTaskStartDurationExpression"
4707 type="tTaskOperation" />
4708     <xsd:element name="setTaskCompletionDeadlineExpression"
4709 type="tTaskOperation" />
4710     <xsd:element name="setTaskCompletionDurationExpression"
4711 type="tTaskOperation" />
4712     <xsd:element name="activate" type="tTaskOperation" />
4713     <xsd:element name="nominate" type="tTaskOperation" />
4714     <xsd:element name="setGenericHumanRole" type="tTaskOperation" />
4715     <xsd:any namespace="##other" processContents="lax" />
4716   </xsd:choice>
4717 </xsd:complexType>
4718 <xsd:complexType name="tTaskOperation">
4719   <xsd:complexContent>
4720     <xsd:restriction base="xsd:anyType" />
4721   </xsd:complexContent>
4722 </xsd:complexType>
4723
4724 <!-- data types for advanced query operations -->
4725 <xsd:element name="taskQueryResultSet" type="tTaskQueryResultSet" />
4726 <xsd:complexType name="tTaskQueryResultSet">
4727   <xsd:sequence>
4728     <xsd:element name="row" type="tTaskQueryResultRow" minOccurs="0"
4729 maxOccurs="unbounded" />
4730   </xsd:sequence>
4731 </xsd:complexType>
4732 <xsd:complexType name="tTaskQueryResultRow">
4733   <xsd:choice minOccurs="0" maxOccurs="unbounded">
4734     <xsd:element name="id" type="xsd:string" />
4735     <xsd:element name="taskType" type="xsd:string" />
4736     <xsd:element name="name" type="xsd:QName" />
4737     <xsd:element name="status" type="tStatus" />
4738     <xsd:element name="priority" type="tPriority" />
4739     <xsd:element name="taskInitiator" type="tOrganizationalEntity" />
4740     <xsd:element name="taskStakeholders" type="tOrganizationalEntity" />
4741     <xsd:element name="potentialOwners" type="tOrganizationalEntity" />
4742     <xsd:element name="businessAdministrators"
4743 type="tOrganizationalEntity" />
4744     <xsd:element name="actualOwner" type="tUser" />
4745     <xsd:element name="notificationRecipients"
4746 type="tOrganizationalEntity" />
4747     <xsd:element name="createdTime" type="xsd:dateTime" />

```



```

4748     <xsd:element name="createdBy" type="xsd:string" />
4749     <xsd:element name="lastModifiedTime" type="xsd:dateTime" />
4750     <xsd:element name="lastModifiedBy" type="xsd:string" />
4751     <xsd:element name="activationTime" type="xsd:dateTime" />
4752     <xsd:element name="expirationTime" type="xsd:dateTime" />
4753     <xsd:element name="isSkipable" type="xsd:boolean" />
4754     <xsd:element name="hasPotentialOwners" type="xsd:boolean" />
4755     <xsd:element name="startByTime" type="xsd:dateTime" />
4756     <xsd:element name="completeByTime" type="xsd:dateTime" />
4757     <xsd:element name="presentationName" type="tPresentationName" />
4758     <xsd:element name="presentationSubject" type="tPresentationSubject" />
4759     <xsd:element name="renderingMethodName" type="xsd:QName" />
4760     <xsd:element name="hasOutput" type="xsd:boolean" />
4761     <xsd:element name="hasFault" type="xsd:boolean" />
4762     <xsd:element name="hasAttachments" type="xsd:boolean" />
4763     <xsd:element name="hasComments" type="xsd:boolean" />
4764     <xsd:element name="escalated" type="xsd:boolean" />
4765     <xsd:element name="parentTaskId" type="xsd:string" />
4766     <xsd:element name="hasSubtasks" type="xsd:boolean" />
4767     <xsd:element name="searchBy" type="xsd:string" />
4768     <xsd:element name="outcome" type="xsd:string" />
4769     <xsd:element name="taskOperations" type="tTaskOperations" />
4770     <xsd:any namespace="##other" processContents="lax" />
4771   </xsd:choice>
4772 </xsd:complexType>
4773 <xsd:complexType name="tFault">
4774   <xsd:sequence>
4775     <xsd:element name="faultName" type="xsd:NCName" />
4776     <xsd:element name="faultData" type="xsd:anyType" />
4777   </xsd:sequence>
4778 </xsd:complexType>
4779
4780 <!-- elements and types for organizational entities -->
4781 <xsd:element name="organizationalEntity" type="tOrganizationalEntity" />
4782 <xsd:complexType name="tOrganizationalEntity">
4783   <xsd:choice maxOccurs="unbounded">
4784     <xsd:element name="user" type="tUser" />
4785     <xsd:element name="group" type="tGroup" />
4786   </xsd:choice>
4787 </xsd:complexType>
4788 <xsd:element name="user" type="tUser" />
4789 <xsd:simpleType name="tUser">
4790   <xsd:restriction base="xsd:string" />
4791 </xsd:simpleType>
4792 <xsd:element name="group" type="tGroup" />
4793 <xsd:simpleType name="tGroup">
4794   <xsd:restriction base="xsd:string" />
4795 </xsd:simpleType>
4796
4797 <!-- input or output message part data -->
4798 <xsd:element name="part" type="tPart" />
4799 <xsd:complexType name="tPart" mixed="true">
4800   <xsd:sequence>
4801     <xsd:any processContents="skip" minOccurs="0" />
4802   </xsd:sequence>
4803   <xsd:attribute name="name" type="xsd:NCName" use="required" />
4804 </xsd:complexType>
4805

```

```

4806 <!-- type container element for one or more message parts -->
4807 <xsd:complexType name="tMessagePartsData">
4808   <xsd:sequence>
4809     <xsd:element ref="part" minOccurs="0" maxOccurs="unbounded" />
4810   </xsd:sequence>
4811 </xsd:complexType>
4812 <xsd:complexType name="tFaultData">
4813   <xsd:sequence>
4814     <xsd:element name="faultName" type="xsd:NCName" />
4815     <xsd:element name="faultData" type="xsd:anyType" />
4816   </xsd:sequence>
4817 </xsd:complexType>
4818 <xsd:element name="attachmentInfos" type="tAttachmentInfos" />
4819 <xsd:complexType name="tAttachmentInfos">
4820   <xsd:sequence>
4821     <xsd:element name="info" type="tAttachmentInfo" minOccurs="0"
4822 maxOccurs="unbounded" />
4823   </xsd:sequence>
4824 </xsd:complexType>
4825 <xsd:element name="comments" type="tComments" />
4826 <xsd:complexType name="tComments">
4827   <xsd:sequence>
4828     <xsd:element ref="comment" minOccurs="0" maxOccurs="unbounded" />
4829   </xsd:sequence>
4830 </xsd:complexType>
4831 <xsd:element name="renderingType" type="xsd:QName" />
4832 <xsd:complexType name="tRenderingTypes">
4833   <xsd:sequence>
4834     <xsd:element ref="renderingType" minOccurs="0" maxOccurs="unbounded" />
4835   </xsd:sequence>
4836 </xsd:complexType>
4837
4838 <!-- Single rendering element that contains rendering type (attribute) and
4839 data. -->
4840 <xsd:element name="rendering" type="tRendering" />
4841 <xsd:complexType name="tRendering">
4842   <xsd:sequence>
4843     <xsd:any namespace="##other" processContents="lax" minOccurs="0"
4844 maxOccurs="unbounded" />
4845   </xsd:sequence>
4846   <xsd:attribute name="type" type="xsd:QName" use="required" />
4847 </xsd:complexType>
4848 <xsd:element name="renderings">
4849   <xsd:complexType>
4850     <xsd:sequence>
4851       <xsd:element ref="rendering" minOccurs="0" maxOccurs="unbounded" />
4852     </xsd:sequence>
4853   </xsd:complexType>
4854 </xsd:element>
4855 <xsd:element name="description" type="xsd:string" />
4856 <xsd:complexType name="tTaskInstanceData">
4857   <xsd:sequence>
4858     <!-- taskDetails contains task ID, meta data, presentation name and
4859 presentation subject. -->
4860     <xsd:element ref="taskDetails" />
4861     <xsd:element ref="description" />
4862     <xsd:element name="input" type="tMessagePartsData" />
4863     <xsd:element name="output" type="tMessagePartsData" nillable="true" />

```

```

4864     <xsd:element name="fault" type="tFaultData" nillable="true"
4865 minOccurs="0" />
4866     <xsd:element ref="renderings" minOccurs="0" />
4867     <xsd:element ref="comments" minOccurs="0" />
4868     <xsd:element ref="attachmentInfos" minOccurs="0" />
4869     <xsd:any namespace="##other" processContents="lax" minOccurs="0"
4870 maxOccurs="unbounded" />
4871   </xsd:sequence>
4872 </xsd:complexType>
4873
4874 <!-- Defines the human task event types -->
4875 <xsd:simpleType name="tTaskEventType">
4876   <xsd:restriction base="xsd:string">
4877     <xsd:enumeration value="create" />
4878     <xsd:enumeration value="claim" />
4879     <xsd:enumeration value="start" />
4880     <xsd:enumeration value="stop" />
4881     <xsd:enumeration value="release" />
4882     <xsd:enumeration value="suspend" />
4883     <xsd:enumeration value="suspendUntil" />
4884     <xsd:enumeration value="resume" />
4885     <xsd:enumeration value="complete" />
4886     <xsd:enumeration value="remove" />
4887     <xsd:enumeration value="fail" />
4888     <xsd:enumeration value="setPriority" />
4889     <xsd:enumeration value="addAttachment" />
4890     <xsd:enumeration value="deleteattachment" />
4891     <xsd:enumeration value="addComment" />
4892     <xsd:enumeration value="skip" />
4893     <xsd:enumeration value="forward" />
4894     <xsd:enumeration value="delegate" />
4895     <xsd:enumeration value="setOutput" />
4896     <xsd:enumeration value="deleteOutput" />
4897     <xsd:enumeration value="setFault" />
4898     <xsd:enumeration value="deleteFault" />
4899     <xsd:enumeration value="activate" />
4900     <xsd:enumeration value="nominate" />
4901     <xsd:enumeration value="setGenericHumanRole" />
4902     <xsd:enumeration value="expire" />
4903     <xsd:enumeration value="escalated" />
4904   </xsd:restriction>
4905 </xsd:simpleType>
4906 <xsd:element name="taskEvent">
4907   <xsd:complexType>
4908     <xsd:annotation>
4909       <xsd:documentation>
4910         A detailed event that represents a change in the task's state.
4911       </xsd:documentation>
4912     </xsd:annotation>
4913     <xsd:sequence>
4914       <!-- event id - unique per task -->
4915       <xsd:element name="id" type="xsd:integer" />
4916       <!-- event date time -->
4917       <xsd:element name="eventTime" type="xsd:dateTime" />
4918       <!-- task ID -->
4919       <xsd:element name="identifier" type="xsd:anyURI" />
4920       <xsd:element name="principal" type="xsd:string" nillable="true"
4921 minOccurs="0" />

```

```

4922     <!-- Event type. Note - using a restricted type limits extensibility
4923 to add custom event types. -->
4924     <xsd:element name="eventType" type="tTaskEventType"/>
4925     <!-- actual owner of the task before the event -->
4926     <xsd:element name="startOwner" type="xsd:string" nillable="true"
4927 minOccurs="0"/>
4928     <!-- actual owner of the task after the event -->
4929     <xsd:element name="endOwner" type="xsd:string" nillable="true"
4930 minOccurs="0"/>
4931     <!-- WSHT task status -->
4932     <xsd:element name="status" type="tStatus"/>
4933     <!-- boolean to indicate this event has optional data -->
4934     <xsd:element name="hasData" type="xsd:boolean" minOccurs="0"/>
4935     <xsd:element name="eventData" type="xsd:anyType" nillable="true"
4936 minOccurs="0"/>
4937     <xsd:element name="faultName" type="xsd:string" nillable="true"
4938 minOccurs="0"/>
4939     <!-- extensibility -->
4940     <xsd:any namespace="##other" processContents="lax" minOccurs="0"
4941 maxOccurs="unbounded"/>
4942     </xsd:sequence>
4943     </xsd:complexType>
4944     </xsd:element>
4945     <!-- Filter allow list event by eventId or other params such as status and
4946 event type -->
4947     <xsd:complexType name="tTaskHistoryFilter">
4948     <xsd:choice>
4949     <xsd:element name="eventId" type="xsd:integer"/>
4950     <!-- Filter to allow narrow down query by status, principal, event
4951 Type. -->
4952     <xsd:sequence>
4953     <xsd:element name="status" type="tStatus" minOccurs="0"
4954 maxOccurs="unbounded"/>
4955     <xsd:element name="eventType" type="tTaskEventType" minOccurs="0"
4956 maxOccurs="unbounded"/>
4957     <xsd:element name="principal" type="xsd:string" minOccurs="0"/>
4958     <xsd:element name="afterEventTime" type="xsd:dateTime"
4959 minOccurs="0"/>
4960     <xsd:element name="beforeEventTime" type="xsd:dateTime"
4961 minOccurs="0"/>
4962     </xsd:sequence>
4963     </xsd:choice>
4964     </xsd:complexType>
4965 </xsd:schema>

```

4966

D. WS-HumanTask Client API Port Type

```
4967 <?xml version="1.0" encoding="UTF-8"?>
4968 <!--
4969 Copyright (c) OASIS Open 2009. All Rights Reserved.
4970 -->
4971 <wsdl:definitions
4972   targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
4973 humantask/api/200803"
4974   xmlns="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/api/200803"
4975   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
4976   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
4977   xmlns:htt="http://docs.oasis-open.org/ns/bpel4people/ws-
4978 humantask/types/200803">
4979
4980   <wsdl:documentation>
4981     Web Service Definition for WS-HumanTask 1.1 - Operations for Client
4982 Applications
4983   </wsdl:documentation>
4984
4985   <wsdl:types>
4986     <xsd:schema
4987       targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
4988 humantask/api/200803"
4989       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
4990       xmlns:htt="http://docs.oasis-open.org/ns/bpel4people/ws-
4991 humantask/types/200803"
4992       elementFormDefault="qualified"
4993       blockDefault="#all">
4994
4995       <xsd:import
4996         namespace="http://docs.oasis-open.org/ns/bpel4people/ws-
4997 humantask/types/200803"
4998         schemaLocation="ws-humantask-types.xsd" />
4999
5000       <!-- Input and output elements -->
5001       <xsd:element name="claim">
5002         <xsd:complexType>
5003           <xsd:sequence>
5004             <xsd:element name="identifier" type="xsd:anyURI" />
5005           </xsd:sequence>
5006         </xsd:complexType>
5007       </xsd:element>
5008       <xsd:element name="claimResponse">
5009         <xsd:complexType>
5010           <xsd:sequence>
5011             <xsd:annotation>
5012               <xsd:documentation>Empty message</xsd:documentation>
5013             </xsd:annotation>
5014           </xsd:sequence>
5015         </xsd:complexType>
5016       </xsd:element>
5017
5018       <xsd:element name="batchClaim">
5019         <xsd:complexType>
```

```

5020     <xsd:sequence>
5021         <xsd:element name="identifier" type="xsd:anyURI"
5022 maxOccurs="unbounded" />
5023     </xsd:sequence>
5024 </xsd:complexType>
5025 </xsd:element>
5026 <xsd:element name="batchClaimResponse">
5027     <xsd:complexType>
5028     <xsd:sequence>
5029         <xsd:element name="batchResponse" type="tBatchResponse"
5030 minOccurs="0" maxOccurs="unbounded" />
5031     </xsd:sequence>
5032 </xsd:complexType>
5033 </xsd:element>
5034
5035 <xsd:element name="start">
5036     <xsd:complexType>
5037     <xsd:sequence>
5038         <xsd:element name="identifier" type="xsd:anyURI" />
5039     </xsd:sequence>
5040 </xsd:complexType>
5041 </xsd:element>
5042 <xsd:element name="startResponse">
5043     <xsd:complexType>
5044     <xsd:sequence>
5045         <xsd:annotation>
5046             <xsd:documentation>Empty message</xsd:documentation>
5047         </xsd:annotation>
5048     </xsd:sequence>
5049 </xsd:complexType>
5050 </xsd:element>
5051
5052 <xsd:element name="batchStart">
5053     <xsd:complexType>
5054     <xsd:sequence>
5055         <xsd:element name="identifier" type="xsd:anyURI"
5056 maxOccurs="unbounded" />
5057     </xsd:sequence>
5058 </xsd:complexType>
5059 </xsd:element>
5060 <xsd:element name="batchStartResponse">
5061     <xsd:complexType>
5062     <xsd:sequence>
5063         <xsd:element name="batchResponse" type="tBatchResponse"
5064 minOccurs="0" maxOccurs="unbounded" />
5065     </xsd:sequence>
5066 </xsd:complexType>
5067 </xsd:element>
5068
5069 <xsd:element name="stop">
5070     <xsd:complexType>
5071     <xsd:sequence>
5072         <xsd:element name="identifier" type="xsd:anyURI" />
5073     </xsd:sequence>
5074 </xsd:complexType>
5075 </xsd:element>
5076 <xsd:element name="stopResponse">
5077     <xsd:complexType>

```

```

5078     <xsd:sequence>
5079         <xsd:annotation>
5080             <xsd:documentation>Empty message</xsd:documentation>
5081         </xsd:annotation>
5082     </xsd:sequence>
5083 </xsd:complexType>
5084 </xsd:element>
5085
5086 <xsd:element name="batchStop">
5087     <xsd:complexType>
5088         <xsd:sequence>
5089             <xsd:element name="identifier" type="xsd:anyURI"
5090 maxOccurs="unbounded" />
5091         </xsd:sequence>
5092     </xsd:complexType>
5093 </xsd:element>
5094 <xsd:element name="batchStopResponse">
5095     <xsd:complexType>
5096         <xsd:sequence>
5097             <xsd:element name="batchResponse" type="tBatchResponse"
5098 minOccurs="0" maxOccurs="unbounded" />
5099         </xsd:sequence>
5100     </xsd:complexType>
5101 </xsd:element>
5102
5103 <xsd:element name="release">
5104     <xsd:complexType>
5105         <xsd:sequence>
5106             <xsd:element name="identifier" type="xsd:anyURI" />
5107         </xsd:sequence>
5108     </xsd:complexType>
5109 </xsd:element>
5110 <xsd:element name="releaseResponse">
5111     <xsd:complexType>
5112         <xsd:sequence>
5113             <xsd:annotation>
5114                 <xsd:documentation>Empty message</xsd:documentation>
5115             </xsd:annotation>
5116         </xsd:sequence>
5117     </xsd:complexType>
5118 </xsd:element>
5119
5120 <xsd:element name="batchRelease">
5121     <xsd:complexType>
5122         <xsd:sequence>
5123             <xsd:element name="identifier" type="xsd:anyURI"
5124 maxOccurs="unbounded" />
5125         </xsd:sequence>
5126     </xsd:complexType>
5127 </xsd:element>
5128 <xsd:element name="batchReleaseResponse">
5129     <xsd:complexType>
5130         <xsd:sequence>
5131             <xsd:element name="batchResponse" type="tBatchResponse"
5132 minOccurs="0" maxOccurs="unbounded" />
5133         </xsd:sequence>
5134     </xsd:complexType>
5135 </xsd:element>

```



```

5136
5137     <xsd:element name="suspend">
5138         <xsd:complexType>
5139             <xsd:sequence>
5140                 <xsd:element name="identifier" type="xsd:anyURI"/>
5141             </xsd:sequence>
5142         </xsd:complexType>
5143     </xsd:element>
5144     <xsd:element name="suspendResponse">
5145         <xsd:complexType>
5146             <xsd:sequence>
5147                 <xsd:annotation>
5148                     <xsd:documentation>Empty message</xsd:documentation>
5149                 </xsd:annotation>
5150             </xsd:sequence>
5151         </xsd:complexType>
5152     </xsd:element>
5153
5154     <xsd:element name="batchSuspend">
5155         <xsd:complexType>
5156             <xsd:sequence>
5157                 <xsd:element name="identifier" type="xsd:anyURI"
5158 maxOccurs="unbounded"/>
5159             </xsd:sequence>
5160         </xsd:complexType>
5161     </xsd:element>
5162     <xsd:element name="batchSuspendResponse">
5163         <xsd:complexType>
5164             <xsd:sequence>
5165                 <xsd:element name="batchResponse" type="tBatchResponse"
5166 minOccurs="0" maxOccurs="unbounded"/>
5167             </xsd:sequence>
5168         </xsd:complexType>
5169     </xsd:element>
5170
5171     <xsd:element name="suspendUntil">
5172         <xsd:complexType>
5173             <xsd:sequence>
5174                 <xsd:element name="identifier" type="xsd:anyURI"/>
5175                 <xsd:element name="time" type="htt:tTime"/>
5176             </xsd:sequence>
5177         </xsd:complexType>
5178     </xsd:element>
5179     <xsd:element name="suspendUntilResponse">
5180         <xsd:complexType>
5181             <xsd:sequence>
5182                 <xsd:annotation>
5183                     <xsd:documentation>Empty message</xsd:documentation>
5184                 </xsd:annotation>
5185             </xsd:sequence>
5186         </xsd:complexType>
5187     </xsd:element>
5188
5189     <xsd:element name="batchSuspendUntil">
5190         <xsd:complexType>
5191             <xsd:sequence>
5192                 <xsd:element name="identifier" type="xsd:anyURI"
5193 maxOccurs="unbounded"/>

```



```

5194         <xsd:element name="time" type="htt:tTime"/>
5195     </xsd:sequence>
5196 </xsd:complexType>
5197 </xsd:element>
5198 <xsd:element name="batchSuspendUntilResponse">
5199     <xsd:complexType>
5200     <xsd:sequence>
5201         <xsd:element name="batchResponse" type="tBatchResponse"
5202 minOccurs="0" maxOccurs="unbounded" />
5203     </xsd:sequence>
5204 </xsd:complexType>
5205 </xsd:element>
5206
5207 <xsd:element name="resume">
5208     <xsd:complexType>
5209     <xsd:sequence>
5210         <xsd:element name="identifier" type="xsd:anyURI"/>
5211     </xsd:sequence>
5212 </xsd:complexType>
5213 </xsd:element>
5214 <xsd:element name="resumeResponse">
5215     <xsd:complexType>
5216     <xsd:sequence>
5217         <xsd:annotation>
5218             <xsd:documentation>Empty message</xsd:documentation>
5219         </xsd:annotation>
5220     </xsd:sequence>
5221 </xsd:complexType>
5222 </xsd:element>
5223
5224 <xsd:element name="batchResume">
5225     <xsd:complexType>
5226     <xsd:sequence>
5227         <xsd:element name="identifier" type="xsd:anyURI"
5228 maxOccurs="unbounded" />
5229     </xsd:sequence>
5230 </xsd:complexType>
5231 </xsd:element>
5232 <xsd:element name="batchResumeResponse">
5233     <xsd:complexType>
5234     <xsd:sequence>
5235         <xsd:element name="batchResponse" type="tBatchResponse"
5236 minOccurs="0" maxOccurs="unbounded" />
5237     </xsd:sequence>
5238 </xsd:complexType>
5239 </xsd:element>
5240
5241 <xsd:element name="complete">
5242     <xsd:complexType>
5243     <xsd:sequence>
5244         <xsd:element name="identifier" type="xsd:anyURI"/>
5245         <xsd:element name="taskData" type="xsd:anyType" minOccurs="0"/>
5246     </xsd:sequence>
5247 </xsd:complexType>
5248 </xsd:element>
5249 <xsd:element name="completeResponse">
5250     <xsd:complexType>
5251     <xsd:sequence>

```

```

5252         <xsd:annotation>
5253             <xsd:documentation>Empty message</xsd:documentation>
5254         </xsd:annotation>
5255     </xsd:sequence>
5256 </xsd:complexType>
5257 </xsd:element>
5258
5259     <xsd:element name="batchComplete">
5260         <xsd:complexType>
5261             <xsd:sequence>
5262                 <xsd:element name="identifier" type="xsd:anyURI"
5263 maxOccurs="unbounded" />
5264             </xsd:sequence>
5265         </xsd:complexType>
5266     </xsd:element>
5267     <xsd:element name="batchCompleteResponse">
5268         <xsd:complexType>
5269             <xsd:sequence>
5270                 <xsd:element name="batchResponse" type="tBatchResponse"
5271 minOccurs="0" maxOccurs="unbounded" />
5272             </xsd:sequence>
5273         </xsd:complexType>
5274     </xsd:element>
5275
5276     <xsd:element name="remove">
5277         <xsd:complexType>
5278             <xsd:sequence>
5279                 <xsd:element name="identifier" type="xsd:anyURI" />
5280             </xsd:sequence>
5281         </xsd:complexType>
5282     </xsd:element>
5283     <xsd:element name="removeResponse">
5284         <xsd:complexType>
5285             <xsd:sequence>
5286                 <xsd:annotation>
5287                     <xsd:documentation>Empty message</xsd:documentation>
5288                 </xsd:annotation>
5289             </xsd:sequence>
5290         </xsd:complexType>
5291     </xsd:element>
5292
5293     <xsd:element name="batchRemove">
5294         <xsd:complexType>
5295             <xsd:sequence>
5296                 <xsd:element name="identifier" type="xsd:anyURI"
5297 maxOccurs="unbounded" />
5298             </xsd:sequence>
5299         </xsd:complexType>
5300     </xsd:element>
5301     <xsd:element name="batchRemoveResponse">
5302         <xsd:complexType>
5303             <xsd:sequence>
5304                 <xsd:element name="batchResponse" type="tBatchResponse"
5305 minOccurs="0" maxOccurs="unbounded" />
5306             </xsd:sequence>
5307         </xsd:complexType>
5308     </xsd:element>
5309

```

```

5310 <xsd:element name="fail">
5311   <xsd:complexType>
5312     <xsd:sequence>
5313       <xsd:element name="identifier" type="xsd:anyURI"/>
5314       <xsd:element name="fault" type="htt:tFault" minOccurs="0"/>
5315     </xsd:sequence>
5316   </xsd:complexType>
5317 </xsd:element>
5318 <xsd:element name="failResponse">
5319   <xsd:complexType>
5320     <xsd:sequence>
5321       <xsd:annotation>
5322         <xsd:documentation>Empty message</xsd:documentation>
5323       </xsd:annotation>
5324     </xsd:sequence>
5325   </xsd:complexType>
5326 </xsd:element>
5327
5328 <xsd:element name="batchFail">
5329   <xsd:complexType>
5330     <xsd:sequence>
5331       <xsd:element name="identifier" type="xsd:anyURI"
5332 maxOccurs="unbounded"/>
5333     </xsd:sequence>
5334   </xsd:complexType>
5335 </xsd:element>
5336 <xsd:element name="batchFailResponse">
5337   <xsd:complexType>
5338     <xsd:sequence>
5339       <xsd:element name="batchResponse" type="tBatchResponse"
5340 minOccurs="0" maxOccurs="unbounded"/>
5341     </xsd:sequence>
5342   </xsd:complexType>
5343 </xsd:element>
5344
5345 <xsd:element name="setPriority">
5346   <xsd:complexType>
5347     <xsd:sequence>
5348       <xsd:element name="identifier" type="xsd:anyURI"/>
5349       <xsd:element name="priority" type="htt:tPriority"/>
5350     </xsd:sequence>
5351   </xsd:complexType>
5352 </xsd:element>
5353 <xsd:element name="setPriorityResponse">
5354   <xsd:complexType>
5355     <xsd:sequence>
5356       <xsd:annotation>
5357         <xsd:documentation>Empty message</xsd:documentation>
5358       </xsd:annotation>
5359     </xsd:sequence>
5360   </xsd:complexType>
5361 </xsd:element>
5362
5363 <xsd:element name="batchSetPriority">
5364   <xsd:complexType>
5365     <xsd:sequence>
5366       <xsd:element name="identifier" type="xsd:anyURI"
5367 maxOccurs="unbounded"/>

```

```

5368         <xsd:element name="priority" type="htt:tPriority" />
5369     </xsd:sequence>
5370 </xsd:complexType>
5371 </xsd:element>
5372 <xsd:element name="batchSetPriorityResponse">
5373     <xsd:complexType>
5374         <xsd:sequence>
5375             <xsd:element name="batchResponse" type="tBatchResponse"
5376 minOccurs="0" maxOccurs="unbounded" />
5377         </xsd:sequence>
5378     </xsd:complexType>
5379 </xsd:element>
5380
5381 <xsd:element name="addAttachment">
5382     <xsd:complexType>
5383         <xsd:sequence>
5384             <xsd:element name="taskIdentifier" type="xsd:anyURI" />
5385             <xsd:element name="name" type="xsd:string" />
5386             <xsd:element name="accessType" type="xsd:string" />
5387             <xsd:element name="contentType" type="xsd:string" />
5388             <xsd:element name="attachment" type="xsd:anyType" />
5389         </xsd:sequence>
5390     </xsd:complexType>
5391 </xsd:element>
5392 <xsd:element name="addAttachmentResponse">
5393     <xsd:complexType>
5394         <xsd:sequence>
5395             <xsd:element name="identifier" type="xsd:anyURI" />
5396         </xsd:sequence>
5397     </xsd:complexType>
5398 </xsd:element>
5399
5400 <xsd:element name="getAttachmentInfos">
5401     <xsd:complexType>
5402         <xsd:sequence>
5403             <xsd:element name="identifier" type="xsd:anyURI" />
5404         </xsd:sequence>
5405     </xsd:complexType>
5406 </xsd:element>
5407 <xsd:element name="getAttachmentInfosResponse">
5408     <xsd:complexType>
5409         <xsd:sequence>
5410             <xsd:element name="info" type="htt:tAttachmentInfo" minOccurs="0"
5411 maxOccurs="unbounded" />
5412         </xsd:sequence>
5413     </xsd:complexType>
5414 </xsd:element>
5415
5416 <xsd:element name="getAttachment">
5417     <xsd:complexType>
5418         <xsd:sequence>
5419             <xsd:element name="taskIdentifier" type="xsd:anyURI" />
5420             <xsd:element name="attachmentIdentifier" type="xsd:anyURI" />
5421         </xsd:sequence>
5422     </xsd:complexType>
5423 </xsd:element>
5424 <xsd:element name="getAttachmentResponse">
5425     <xsd:complexType>

```

```

5426     <xsd:sequence>
5427         <xsd:element name="attachment" type="htt:tAttachment"
5428 minOccurs="0" maxOccurs="unbounded" />
5429     </xsd:sequence>
5430 </xsd:complexType>
5431 </xsd:element>
5432
5433 <xsd:element name="deleteAttachment">
5434     <xsd:complexType>
5435         <xsd:sequence>
5436             <xsd:element name="taskIdIdentifier" type="xsd:anyURI" />
5437             <xsd:element name="attachmentIdentifier" type="xsd:anyURI" />
5438         </xsd:sequence>
5439     </xsd:complexType>
5440 </xsd:element>
5441 <xsd:element name="deleteAttachmentResponse">
5442     <xsd:complexType>
5443         <xsd:sequence>
5444             <xsd:annotation>
5445                 <xsd:documentation>Empty message</xsd:documentation>
5446             </xsd:annotation>
5447         </xsd:sequence>
5448     </xsd:complexType>
5449 </xsd:element>
5450
5451 <xsd:element name="addComment">
5452     <xsd:complexType>
5453         <xsd:sequence>
5454             <xsd:element name="identifier" type="xsd:anyURI" />
5455             <xsd:element name="text" type="xsd:string" />
5456         </xsd:sequence>
5457     </xsd:complexType>
5458 </xsd:element>
5459 <xsd:element name="addCommentResponse">
5460     <xsd:complexType>
5461         <xsd:sequence>
5462             <xsd:element name="commentID" type="xsd:string" />
5463         </xsd:sequence>
5464     </xsd:complexType>
5465 </xsd:element>
5466
5467 <xsd:element name="updateComment">
5468     <xsd:complexType>
5469         <xsd:sequence>
5470             <xsd:element name="taskIdIdentifier" type="xsd:anyURI" />
5471             <xsd:element name="commentIdentifier" type="xsd:anyURI" />
5472             <xsd:element name="text" type="xsd:string" />
5473         </xsd:sequence>
5474     </xsd:complexType>
5475 </xsd:element>
5476 <xsd:element name="updateCommentResponse">
5477     <xsd:complexType>
5478         <xsd:sequence>
5479             <xsd:annotation>
5480                 <xsd:documentation>Empty message</xsd:documentation>
5481             </xsd:annotation>
5482         </xsd:sequence>
5483     </xsd:complexType>

```

```

5484 </xsd:element>
5485
5486 <xsd:element name="deleteComment">
5487   <xsd:complexType>
5488     <xsd:sequence>
5489       <xsd:element name="taskIdIdentifier" type="xsd:anyURI"/>
5490       <xsd:element name="commentIdentifier" type="xsd:anyURI"/>
5491     </xsd:sequence>
5492   </xsd:complexType>
5493 </xsd:element>
5494 <xsd:element name="deleteCommentResponse">
5495   <xsd:complexType>
5496     <xsd:sequence>
5497       <xsd:annotation>
5498         <xsd:documentation>Empty message</xsd:documentation>
5499       </xsd:annotation>
5500     </xsd:sequence>
5501   </xsd:complexType>
5502 </xsd:element>
5503
5504 <xsd:element name="getComments">
5505   <xsd:complexType>
5506     <xsd:sequence>
5507       <xsd:element name="identifier" type="xsd:anyURI"/>
5508     </xsd:sequence>
5509   </xsd:complexType>
5510 </xsd:element>
5511 <xsd:element name="getCommentsResponse">
5512   <xsd:complexType>
5513     <xsd:sequence>
5514       <xsd:element name="comment" type="htt:tComment" minOccurs="0"
5515 maxOccurs="unbounded"/>
5516     </xsd:sequence>
5517   </xsd:complexType>
5518 </xsd:element>
5519
5520 <xsd:element name="skip">
5521   <xsd:complexType>
5522     <xsd:sequence>
5523       <xsd:element name="identifier" type="xsd:anyURI"/>
5524     </xsd:sequence>
5525   </xsd:complexType>
5526 </xsd:element>
5527 <xsd:element name="skipResponse">
5528   <xsd:complexType>
5529     <xsd:sequence>
5530       <xsd:annotation>
5531         <xsd:documentation>Empty message</xsd:documentation>
5532       </xsd:annotation>
5533     </xsd:sequence>
5534   </xsd:complexType>
5535 </xsd:element>
5536
5537 <xsd:element name="batchSkip">
5538   <xsd:complexType>
5539     <xsd:sequence>
5540       <xsd:element name="identifier" type="xsd:anyURI"
5541 maxOccurs="unbounded"/>

```

```

5542     </xsd:sequence>
5543     </xsd:complexType>
5544 </xsd:element>
5545 <xsd:element name="batchSkipResponse">
5546     <xsd:complexType>
5547         <xsd:sequence>
5548             <xsd:element name="batchResponse" type="tBatchResponse"
5549 minOccurs="0" maxOccurs="unbounded" />
5550         </xsd:sequence>
5551     </xsd:complexType>
5552 </xsd:element>
5553
5554 <xsd:element name="forward">
5555     <xsd:complexType>
5556         <xsd:sequence>
5557             <xsd:element name="identifier" type="xsd:anyURI" />
5558             <xsd:element name="organizationalEntity"
5559 type="htt:tOrganizationalEntity" />
5560         </xsd:sequence>
5561     </xsd:complexType>
5562 </xsd:element>
5563 <xsd:element name="forwardResponse">
5564     <xsd:complexType>
5565         <xsd:sequence>
5566             <xsd:annotation>
5567                 <xsd:documentation>Empty message</xsd:documentation>
5568             </xsd:annotation>
5569         </xsd:sequence>
5570     </xsd:complexType>
5571 </xsd:element>
5572
5573 <xsd:element name="batchForward">
5574     <xsd:complexType>
5575         <xsd:sequence>
5576             <xsd:element name="identifier" type="xsd:anyURI"
5577 maxOccurs="unbounded" />
5578             <xsd:element name="organizationalEntity"
5579 type="htt:tOrganizationalEntity" />
5580         </xsd:sequence>
5581     </xsd:complexType>
5582 </xsd:element>
5583 <xsd:element name="batchForwardResponse">
5584     <xsd:complexType>
5585         <xsd:sequence>
5586             <xsd:element name="batchResponse" type="tBatchResponse"
5587 minOccurs="0" maxOccurs="unbounded" />
5588         </xsd:sequence>
5589     </xsd:complexType>
5590 </xsd:element>
5591
5592 <xsd:element name="delegate">
5593     <xsd:complexType>
5594         <xsd:sequence>
5595             <xsd:element name="identifier" type="xsd:anyURI" />
5596             <xsd:element name="organizationalEntity"
5597 type="htt:tOrganizationalEntity" />
5598         </xsd:sequence>
5599     </xsd:complexType>

```



```

5600 </xsd:element>
5601 <xsd:element name="delegateResponse">
5602 <xsd:complexType>
5603 <xsd:sequence>
5604 <xsd:annotation>
5605 <xsd:documentation>Empty message</xsd:documentation>
5606 </xsd:annotation>
5607 </xsd:sequence>
5608 </xsd:complexType>
5609 </xsd:element>
5610
5611 <xsd:element name="batchDelegate">
5612 <xsd:complexType>
5613 <xsd:sequence>
5614 <xsd:element name="identifier" type="xsd:anyURI"
5615 maxOccurs="unbounded" />
5616 <xsd:element name="organizationalEntity"
5617 type="htt:tOrganizationalEntity" />
5618 </xsd:sequence>
5619 </xsd:complexType>
5620 </xsd:element>
5621 <xsd:element name="batchDelegateResponse">
5622 <xsd:complexType>
5623 <xsd:sequence>
5624 <xsd:element name="batchResponse" type="tBatchResponse"
5625 minOccurs="0" maxOccurs="unbounded" />
5626 </xsd:sequence>
5627 </xsd:complexType>
5628 </xsd:element>
5629
5630 <xsd:element name="getRendering">
5631 <xsd:complexType>
5632 <xsd:sequence>
5633 <xsd:element name="identifier" type="xsd:anyType" />
5634 <xsd:element name="renderingType" type="xsd:QName" />
5635 </xsd:sequence>
5636 </xsd:complexType>
5637 </xsd:element>
5638 <xsd:element name="getRenderingResponse">
5639 <xsd:complexType>
5640 <xsd:sequence>
5641 <xsd:element name="rendering" type="xsd:anyType" />
5642 </xsd:sequence>
5643 </xsd:complexType>
5644 </xsd:element>
5645
5646 <xsd:element name="getRenderingTypes">
5647 <xsd:complexType>
5648 <xsd:sequence>
5649 <xsd:element name="identifier" type="xsd:anyType" />
5650 </xsd:sequence>
5651 </xsd:complexType>
5652 </xsd:element>
5653 <xsd:element name="getRenderingTypesResponse">
5654 <xsd:complexType>
5655 <xsd:sequence>
5656 <xsd:element name="renderingType" type="xsd:QName" minOccurs="0"
5657 maxOccurs="unbounded" />

```



```

5658     </xsd:sequence>
5659   </xsd:complexType>
5660 </xsd:element>
5661
5662   <xsd:element name="getTaskDetails">
5663     <xsd:complexType>
5664       <xsd:sequence>
5665         <xsd:element name="identifier" type="xsd:anyURI"/>
5666       </xsd:sequence>
5667     </xsd:complexType>
5668   </xsd:element>
5669   <xsd:element name="getTaskDetailsResponse">
5670     <xsd:complexType>
5671       <xsd:sequence>
5672         <xsd:element name="taskDetails" type="htt:tTaskDetails"/>
5673       </xsd:sequence>
5674     </xsd:complexType>
5675   </xsd:element>
5676
5677   <xsd:element name="getTaskDescription">
5678     <xsd:complexType>
5679       <xsd:sequence>
5680         <xsd:element name="identifier" type="xsd:anyURI"/>
5681         <xsd:element name="contentType" type="xsd:string" minOccurs="0"/>
5682       </xsd:sequence>
5683     </xsd:complexType>
5684   </xsd:element>
5685   <xsd:element name="getTaskDescriptionResponse">
5686     <xsd:complexType>
5687       <xsd:sequence>
5688         <xsd:element name="description" type="xsd:string"/>
5689       </xsd:sequence>
5690     </xsd:complexType>
5691   </xsd:element>
5692
5693   <xsd:element name="setOutput">
5694     <xsd:complexType>
5695       <xsd:sequence>
5696         <xsd:element name="identifier" type="xsd:anyURI"/>
5697         <xsd:element name="part" type="xsd:NCName" minOccurs="0"/>
5698         <xsd:element name="taskData" type="xsd:anyType"/>
5699       </xsd:sequence>
5700     </xsd:complexType>
5701   </xsd:element>
5702   <xsd:element name="setOutputResponse">
5703     <xsd:complexType>
5704       <xsd:sequence>
5705         <xsd:annotation>
5706           <xsd:documentation>Empty message</xsd:documentation>
5707         </xsd:annotation>
5708       </xsd:sequence>
5709     </xsd:complexType>
5710   </xsd:element>
5711
5712   <xsd:element name="deleteOutput">
5713     <xsd:complexType>
5714       <xsd:sequence>
5715         <xsd:element name="identifier" type="xsd:anyURI"/>

```

```

5716     </xsd:sequence>
5717   </xsd:complexType>
5718 </xsd:element>
5719 <xsd:element name="deleteOutputResponse">
5720   <xsd:complexType>
5721     <xsd:sequence>
5722       <xsd:annotation>
5723         <xsd:documentation>Empty message</xsd:documentation>
5724       </xsd:annotation>
5725     </xsd:sequence>
5726   </xsd:complexType>
5727 </xsd:element>
5728
5729 <xsd:element name="setFault">
5730   <xsd:complexType>
5731     <xsd:sequence>
5732       <xsd:element name="identifier" type="xsd:anyURI"/>
5733       <xsd:element name="fault" type="http:Fault"/>
5734     </xsd:sequence>
5735   </xsd:complexType>
5736 </xsd:element>
5737 <xsd:element name="setFaultResponse">
5738   <xsd:complexType>
5739     <xsd:sequence>
5740       <xsd:annotation>
5741         <xsd:documentation>Empty message</xsd:documentation>
5742       </xsd:annotation>
5743     </xsd:sequence>
5744   </xsd:complexType>
5745 </xsd:element>
5746
5747 <xsd:element name="deleteFault">
5748   <xsd:complexType>
5749     <xsd:sequence>
5750       <xsd:element name="identifier" type="xsd:anyURI"/>
5751     </xsd:sequence>
5752   </xsd:complexType>
5753 </xsd:element>
5754 <xsd:element name="deleteFaultResponse">
5755   <xsd:complexType>
5756     <xsd:sequence>
5757       <xsd:annotation>
5758         <xsd:documentation>Empty message</xsd:documentation>
5759       </xsd:annotation>
5760     </xsd:sequence>
5761   </xsd:complexType>
5762 </xsd:element>
5763
5764 <xsd:element name="getInput">
5765   <xsd:complexType>
5766     <xsd:sequence>
5767       <xsd:element name="identifier" type="xsd:anyURI"/>
5768       <xsd:element name="part" type="xsd:NCName" minOccurs="0"/>
5769     </xsd:sequence>
5770   </xsd:complexType>
5771 </xsd:element>
5772 <xsd:element name="getInputResponse">
5773   <xsd:complexType>

```

```

5774     <xsd:sequence>
5775         <xsd:element name="taskData" type="xsd:anyType" />
5776     </xsd:sequence>
5777 </xsd:complexType>
5778 </xsd:element>
5779
5780 <xsd:element name="getOutput">
5781     <xsd:complexType>
5782         <xsd:sequence>
5783             <xsd:element name="identifier" type="xsd:anyURI" />
5784             <xsd:element name="part" type="xsd:NCName" minOccurs="0" />
5785         </xsd:sequence>
5786     </xsd:complexType>
5787 </xsd:element>
5788 <xsd:element name="getOutputResponse">
5789     <xsd:complexType>
5790         <xsd:sequence>
5791             <xsd:element name="taskData" type="xsd:anyType" />
5792         </xsd:sequence>
5793     </xsd:complexType>
5794 </xsd:element>
5795
5796 <xsd:element name="getFault">
5797     <xsd:complexType>
5798         <xsd:sequence>
5799             <xsd:element name="identifier" type="xsd:anyURI" />
5800         </xsd:sequence>
5801     </xsd:complexType>
5802 </xsd:element>
5803 <xsd:element name="getFaultResponse">
5804     <xsd:complexType>
5805         <xsd:sequence>
5806             <xsd:element name="fault" type="htt:tFault" />
5807         </xsd:sequence>
5808     </xsd:complexType>
5809 </xsd:element>
5810
5811 <xsd:element name="getMyTaskAbstracts">
5812     <xsd:complexType>
5813         <xsd:sequence>
5814             <xsd:element name="taskType" type="xsd:string" />
5815             <xsd:element name="genericHumanRole" type="xsd:string"
5816 minOccurs="0" />
5817             <xsd:element name="workQueue" type="xsd:string" minOccurs="0" />
5818             <xsd:element name="status" type="htt:tStatus" minOccurs="0"
5819 maxOccurs="unbounded" />
5820             <xsd:element name="whereClause" type="xsd:string" minOccurs="0" />
5821             <xsd:element name="createdOnClause" type="xsd:string"
5822 minOccurs="0" />
5823             <xsd:element name="maxTasks" type="xsd:int" minOccurs="0" />
5824         </xsd:sequence>
5825     </xsd:complexType>
5826 </xsd:element>
5827 <xsd:element name="getMyTaskAbstractsResponse">
5828     <xsd:complexType>
5829         <xsd:sequence>
5830             <xsd:element name="taskAbstract" type="htt:tTaskAbstract"
5831 minOccurs="0" maxOccurs="unbounded" />

```

```

5832     </xsd:sequence>
5833 </xsd:complexType>
5834 </xsd:element>
5835
5836 <xsd:element name="getMyTaskDetails">
5837 <xsd:complexType>
5838 <xsd:sequence>
5839 <xsd:element name="taskType" type="xsd:string"/>
5840 <xsd:element name="genericHumanRole" type="xsd:string"
5841 minOccurs="0"/>
5842 <xsd:element name="workQueue" type="xsd:string" minOccurs="0"/>
5843 <xsd:element name="status" type="htt:tStatus" minOccurs="0"
5844 maxOccurs="unbounded"/>
5845 <xsd:element name="whereClause" type="xsd:string" minOccurs="0"/>
5846 <xsd:element name="createdOnClause" type="xsd:string"
5847 minOccurs="0"/>
5848 <xsd:element name="maxTasks" type="xsd:int" minOccurs="0"/>
5849 </xsd:sequence>
5850 </xsd:complexType>
5851 </xsd:element>
5852 <xsd:element name="getMyTaskDetailsResponse">
5853 <xsd:complexType>
5854 <xsd:sequence>
5855 <xsd:element name="taskDetails" type="htt:tTaskDetails"
5856 minOccurs="0" maxOccurs="unbounded"/>
5857 </xsd:sequence>
5858 </xsd:complexType>
5859 </xsd:element>
5860
5861 <xsd:element name="query">
5862 <xsd:complexType>
5863 <xsd:sequence>
5864 <xsd:element name="selectClause" type="xsd:string"/>
5865 <xsd:element name="whereClause" type="xsd:string" minOccurs="0"/>
5866 <xsd:element name="orderByClause" type="xsd:string"
5867 minOccurs="0"/>
5868 <xsd:element name="maxTasks" type="xsd:int" minOccurs="0"/>
5869 <xsd:element name="taskIndexOffset" type="xsd:int"
5870 minOccurs="0"/>
5871 </xsd:sequence>
5872 </xsd:complexType>
5873 </xsd:element>
5874 <xsd:element name="queryResponse">
5875 <xsd:complexType>
5876 <xsd:sequence>
5877 <xsd:element name="taskQueryResultSet"
5878 type="htt:tTaskQueryResultSet"/>
5879 </xsd:sequence>
5880 </xsd:complexType>
5881 </xsd:element>
5882
5883 <xsd:element name="activate">
5884 <xsd:complexType>
5885 <xsd:sequence>
5886 <xsd:element name="identifier" type="xsd:anyURI"/>
5887 </xsd:sequence>
5888 </xsd:complexType>
5889 </xsd:element>

```

```

5890 <xsd:element name="activateResponse">
5891   <xsd:complexType>
5892     <xsd:sequence>
5893       <xsd:annotation>
5894         <xsd:documentation>Empty message</xsd:documentation>
5895       </xsd:annotation>
5896     </xsd:sequence>
5897   </xsd:complexType>
5898 </xsd:element>
5899
5900 <xsd:element name="batchActivate">
5901   <xsd:complexType>
5902     <xsd:sequence>
5903       <xsd:element name="identifier" type="xsd:anyURI"
5904 maxOccurs="unbounded" />
5905     </xsd:sequence>
5906   </xsd:complexType>
5907 </xsd:element>
5908 <xsd:element name="batchActivateResponse">
5909   <xsd:complexType>
5910     <xsd:sequence>
5911       <xsd:element name="batchResponse" type="tBatchResponse"
5912 minOccurs="0" maxOccurs="unbounded" />
5913     </xsd:sequence>
5914   </xsd:complexType>
5915 </xsd:element>
5916
5917 <xsd:element name="nominate">
5918   <xsd:complexType>
5919     <xsd:sequence>
5920       <xsd:element name="identifier" type="xsd:anyURI" />
5921       <xsd:element name="organizationalEntity"
5922 type="htt:tOrganizationalEntity" />
5923     </xsd:sequence>
5924   </xsd:complexType>
5925 </xsd:element>
5926 <xsd:element name="nominateResponse">
5927   <xsd:complexType>
5928     <xsd:sequence>
5929       <xsd:annotation>
5930         <xsd:documentation>Empty message</xsd:documentation>
5931       </xsd:annotation>
5932     </xsd:sequence>
5933   </xsd:complexType>
5934 </xsd:element>
5935
5936 <xsd:element name="batchNominate">
5937   <xsd:complexType>
5938     <xsd:sequence>
5939       <xsd:element name="identifier" type="xsd:anyURI"
5940 maxOccurs="unbounded" />
5941     </xsd:sequence>
5942   </xsd:complexType>
5943 </xsd:element>
5944 <xsd:element name="batchNominateResponse">
5945   <xsd:complexType>
5946     <xsd:sequence>

```

```

5947         <xsd:element name="batchResponse" type="tBatchResponse"
5948 minOccurs="0" maxOccurs="unbounded" />
5949         </xsd:sequence>
5950     </xsd:complexType>
5951 </xsd:element>
5952
5953     <xsd:element name="setGenericHumanRole">
5954     <xsd:complexType>
5955     <xsd:sequence>
5956         <xsd:element name="identifier" type="xsd:anyURI" />
5957         <xsd:element name="genericHumanRole" type="xsd:string" />
5958         <xsd:element name="organizationalEntity"
5959 type="htt:tOrganizationalEntity" />
5960     </xsd:sequence>
5961     </xsd:complexType>
5962 </xsd:element>
5963     <xsd:element name="setGenericHumanRoleResponse">
5964     <xsd:complexType>
5965     <xsd:sequence>
5966         <xsd:annotation>
5967             <xsd:documentation>Empty message</xsd:documentation>
5968         </xsd:annotation>
5969     </xsd:sequence>
5970     </xsd:complexType>
5971 </xsd:element>
5972
5973     <xsd:element name="batchSetGenericHumanRole">
5974     <xsd:complexType>
5975     <xsd:sequence>
5976         <xsd:element name="identifier" type="xsd:anyURI"
5977 maxOccurs="unbounded" />
5978         <xsd:element name="genericHumanRole" type="xsd:string" />
5979         <xsd:element name="organizationalEntity"
5980 type="htt:tOrganizationalEntity" />
5981     </xsd:sequence>
5982     </xsd:complexType>
5983 </xsd:element>
5984     <xsd:element name="batchSetGenericHumanRoleResponse">
5985     <xsd:complexType>
5986     <xsd:sequence>
5987         <xsd:element name="batchResponse" type="tBatchResponse"
5988 minOccurs="0" maxOccurs="unbounded" />
5989     </xsd:sequence>
5990     </xsd:complexType>
5991 </xsd:element>
5992
5993
5994     <xsd:element name="getOutcome">
5995     <xsd:complexType>
5996     <xsd:sequence>
5997         <xsd:element name="identifier" type="xsd:anyURI" />
5998     </xsd:sequence>
5999     </xsd:complexType>
6000 </xsd:element>
6001     <xsd:element name="getOutcomeResponse">
6002     <xsd:complexType>
6003     <xsd:sequence>
6004         <xsd:element name="outcome" type="xsd:string" />

```

```

6005     </xsd:sequence>
6006   </xsd:complexType>
6007 </xsd:element>
6008
6009   <xsd:element name="getTaskOperations">
6010     <xsd:complexType>
6011       <xsd:sequence>
6012         <xsd:element name="identifier" type="xsd:anyURI"/>
6013       </xsd:sequence>
6014     </xsd:complexType>
6015   </xsd:element>
6016   <xsd:element name="getTaskOperationsResponse">
6017     <xsd:complexType>
6018       <xsd:sequence>
6019         <xsd:element name="taskOperations" type="htt:tTaskOperations"/>
6020       </xsd:sequence>
6021     </xsd:complexType>
6022   </xsd:element>
6023
6024   <xsd:element name="getTaskInstanceData">
6025     <xsd:complexType>
6026       <xsd:sequence>
6027         <xsd:element name="identifier" type="xsd:anyURI"/>
6028         <xsd:element name="properties" type="xsd:string"/>
6029         <xsd:element name="renderingPreferences"
6030 type="htt:tRenderingTypes" minOccurs="0" maxOccurs="unbounded"/>
6031       </xsd:sequence>
6032     </xsd:complexType>
6033   </xsd:element>
6034   <xsd:element name="getTaskInstanceDataResponse">
6035     <xsd:complexType>
6036       <xsd:sequence>
6037         <xsd:element name="taskInstanceData"
6038 type="htt:tTaskInstanceData"/>
6039       </xsd:sequence>
6040     </xsd:complexType>
6041   </xsd:element>
6042
6043   <xsd:element name="getTaskHistory">
6044     <xsd:complexType>
6045       <xsd:sequence>
6046         <xsd:element name="identifier" type="xsd:anyURI"/>
6047         <xsd:element name="filter" type="htt:tTaskHistoryFilter"
6048 minOccurs="0"/>
6049         <xsd:element name="startIndex" type="xsd:integer" minOccurs="0"/>
6050         <xsd:element name="maxTasks" type="xsd:integer" minOccurs="0"/>
6051       </xsd:sequence>
6052       <xsd:attribute name="includeData" type="xsd:boolean"/>
6053     </xsd:complexType>
6054   </xsd:element>
6055   <xsd:element name="getTaskHistoryResponse">
6056     <xsd:complexType>
6057       <xsd:sequence>
6058         <xsd:element name="taskEvent" type="htt:tTaskEventType"
6059 minOccurs="0" maxOccurs="unbounded"/>
6060       </xsd:sequence>
6061     </xsd:complexType>
6062   </xsd:element>

```



```

6063
6064 <xsd:element name="setTaskStartDeadlineExpression">
6065   <xsd:complexType>
6066     <xsd:sequence>
6067       <xsd:element name="identifier" type="xsd:anyURI"/>
6068       <xsd:element name="deadlineName" type="xsd:NCName"/>
6069       <xsd:element name="deadlineExpression" type="xsd:string"/>
6070     </xsd:sequence>
6071   </xsd:complexType>
6072 </xsd:element>
6073 <xsd:element name="setTaskStartDeadlineExpressionResponse">
6074   <xsd:complexType>
6075     <xsd:sequence>
6076       <xsd:annotation>
6077         <xsd:documentation>Empty message</xsd:documentation>
6078       </xsd:annotation>
6079     </xsd:sequence>
6080   </xsd:complexType>
6081 </xsd:element>
6082
6083 <xsd:element name="setTaskStartDurationExpression">
6084   <xsd:complexType>
6085     <xsd:sequence>
6086       <xsd:element name="identifier" type="xsd:anyURI"/>
6087       <xsd:element name="deadlineName" type="xsd:NCName"/>
6088       <xsd:element name="durationExpression" type="xsd:string"/>
6089     </xsd:sequence>
6090   </xsd:complexType>
6091 </xsd:element>
6092 <xsd:element name="setTaskStartDurationExpressionResponse">
6093   <xsd:complexType>
6094     <xsd:sequence>
6095       <xsd:annotation>
6096         <xsd:documentation>Empty message</xsd:documentation>
6097       </xsd:annotation>
6098     </xsd:sequence>
6099   </xsd:complexType>
6100 </xsd:element>
6101
6102 <xsd:element name="setTaskCompletionDeadlineExpression">
6103   <xsd:complexType>
6104     <xsd:sequence>
6105       <xsd:element name="identifier" type="xsd:anyURI"/>
6106       <xsd:element name="deadlineName" type="xsd:NCName"/>
6107       <xsd:element name="deadlineExpression" type="xsd:string"/>
6108     </xsd:sequence>
6109   </xsd:complexType>
6110 </xsd:element>
6111 <xsd:element name="setTaskCompletionDeadlineExpressionResponse">
6112   <xsd:complexType>
6113     <xsd:sequence>
6114       <xsd:annotation>
6115         <xsd:documentation>Empty message</xsd:documentation>
6116       </xsd:annotation>
6117     </xsd:sequence>
6118   </xsd:complexType>
6119 </xsd:element>
6120

```



```

6121 <xsd:element name="setTaskCompletionDurationExpression">
6122   <xsd:complexType>
6123     <xsd:sequence>
6124       <xsd:element name="identifier" type="xsd:anyURI"/>
6125       <xsd:element name="deadlineName" type="xsd:NCName"/>
6126       <xsd:element name="durationExpression" type="xsd:string"/>
6127     </xsd:sequence>
6128   </xsd:complexType>
6129 </xsd:element>
6130 <xsd:element name="setTaskCompletionDurationExpressionResponse">
6131   <xsd:complexType>
6132     <xsd:sequence>
6133       <xsd:annotation>
6134         <xsd:documentation>Empty message</xsd:documentation>
6135       </xsd:annotation>
6136     </xsd:sequence>
6137   </xsd:complexType>
6138 </xsd:element>
6139
6140 <!-- Fault elements -->
6141 <xsd:element name="illegalState">
6142   <xsd:complexType>
6143     <xsd:sequence>
6144       <xsd:element name="status" type="htt:tStatus"/>
6145       <xsd:element name="message" type="xsd:string"/>
6146     </xsd:sequence>
6147   </xsd:complexType>
6148 </xsd:element>
6149
6150 <xsd:element name="illegalArgument" type="xsd:string"/>
6151
6152 <xsd:element name="illegalAccess" type="xsd:string"/>
6153
6154 <xsd:element name="illegalOperation" type="xsd:string"/>
6155
6156 <xsd:element name="recipientNotAllowed" type="xsd:string"/>
6157
6158 <xsd:complexType name="tBatchResponse">
6159   <xsd:sequence>
6160     <xsd:element name="identifier" type="xsd:anyURI"/>
6161     <xsd:choice>
6162       <xsd:element ref="illegalState"/>
6163       <xsd:element ref="illegalArgument"/>
6164       <xsd:element ref="illegalAccess"/>
6165       <xsd:element ref="illegalOperation"/>
6166       <xsd:element ref="recipientNotAllowed"/>
6167       <xsd:any namespace="##other" processContents="lax"/>
6168     </xsd:choice>
6169   </xsd:sequence>
6170 </xsd:complexType>
6171
6172 </xsd:schema>
6173 </wsdl:types>
6174
6175 <!-- Declaration of messages -->
6176 <wsdl:message name="claim">
6177   <wsdl:part name="claim" element="claim"/>
6178 </wsdl:message>

```

```
6179 <wsdl:message name="claimResponse">
6180   <wsdl:part name="claimResponse" element="claimResponse" />
6181 </wsdl:message>
6182
6183 <wsdl:message name="batchClaim">
6184   <wsdl:part name="batchClaim" element="batchClaim" />
6185 </wsdl:message>
6186 <wsdl:message name="batchClaimResponse">
6187   <wsdl:part name="batchClaimResponse" element="batchClaimResponse" />
6188 </wsdl:message>
6189
6190 <wsdl:message name="start">
6191   <wsdl:part name="start" element="start" />
6192 </wsdl:message>
6193 <wsdl:message name="startResponse">
6194   <wsdl:part name="startResponse" element="startResponse" />
6195 </wsdl:message>
6196
6197 <wsdl:message name="batchStart">
6198   <wsdl:part name="batchStart" element="batchStart" />
6199 </wsdl:message>
6200 <wsdl:message name="batchStartResponse">
6201   <wsdl:part name="batchStartResponse" element="batchStartResponse" />
6202 </wsdl:message>
6203
6204 <wsdl:message name="stop">
6205   <wsdl:part name="stop" element="stop" />
6206 </wsdl:message>
6207 <wsdl:message name="stopResponse">
6208   <wsdl:part name="stopResponse" element="stopResponse" />
6209 </wsdl:message>
6210
6211 <wsdl:message name="batchStop">
6212   <wsdl:part name="batchStop" element="batchStop" />
6213 </wsdl:message>
6214 <wsdl:message name="batchStopResponse">
6215   <wsdl:part name="batchStopResponse" element="batchStopResponse" />
6216 </wsdl:message>
6217
6218 <wsdl:message name="release">
6219   <wsdl:part name="release" element="release" />
6220 </wsdl:message>
6221 <wsdl:message name="releaseResponse">
6222   <wsdl:part name="releaseResponse" element="releaseResponse" />
6223 </wsdl:message>
6224
6225 <wsdl:message name="batchRelease">
6226   <wsdl:part name="batchRelease" element="batchRelease" />
6227 </wsdl:message>
6228 <wsdl:message name="batchReleaseResponse">
6229   <wsdl:part name="batchReleaseResponse" element="batchReleaseResponse" />
6230 </wsdl:message>
6231
6232 <wsdl:message name="suspend">
6233   <wsdl:part name="suspend" element="suspend" />
6234 </wsdl:message>
6235 <wsdl:message name="suspendResponse">
6236   <wsdl:part name="suspendResponse" element="suspendResponse" />
```

```

6237 </wsdl:message>
6238
6239 <wsdl:message name="batchSuspend">
6240   <wsdl:part name="batchSuspend" element="batchSuspend"/>
6241 </wsdl:message>
6242 <wsdl:message name="batchSuspendResponse">
6243   <wsdl:part name="batchSuspendResponse" element="batchSuspendResponse"/>
6244 </wsdl:message>
6245
6246 <wsdl:message name="suspendUntil">
6247   <wsdl:part name="suspendUntil" element="suspendUntil"/>
6248 </wsdl:message>
6249 <wsdl:message name="suspendUntilResponse">
6250   <wsdl:part name="suspendUntilResponse" element="suspendUntilResponse"/>
6251 </wsdl:message>
6252
6253 <wsdl:message name="batchSuspendUntil">
6254   <wsdl:part name="batchSuspendUntil" element="batchSuspendUntil"/>
6255 </wsdl:message>
6256 <wsdl:message name="batchSuspendUntilResponse">
6257   <wsdl:part name="batchSuspendUntilResponse"
6258 element="batchSuspendUntilResponse"/>
6259 </wsdl:message>
6260
6261 <wsdl:message name="resume">
6262   <wsdl:part name="resume" element="resume"/>
6263 </wsdl:message>
6264 <wsdl:message name="resumeResponse">
6265   <wsdl:part name="resumeResponse" element="resumeResponse"/>
6266 </wsdl:message>
6267
6268 <wsdl:message name="batchResume">
6269   <wsdl:part name="batchResume" element="batchResume"/>
6270 </wsdl:message>
6271 <wsdl:message name="batchResumeResponse">
6272   <wsdl:part name="batchResumeResponse" element="batchResumeResponse"/>
6273 </wsdl:message>
6274
6275 <wsdl:message name="complete">
6276   <wsdl:part name="complete" element="complete"/>
6277 </wsdl:message>
6278 <wsdl:message name="completeResponse">
6279   <wsdl:part name="completeResponse" element="completeResponse"/>
6280 </wsdl:message>
6281
6282 <wsdl:message name="batchComplete">
6283   <wsdl:part name="batchComplete" element="batchComplete"/>
6284 </wsdl:message>
6285 <wsdl:message name="batchCompleteResponse">
6286   <wsdl:part name="batchCompleteResponse" element="batchCompleteResponse"/>
6287 </wsdl:message>
6288
6289 <wsdl:message name="remove">
6290   <wsdl:part name="remove" element="remove"/>
6291 </wsdl:message>
6292 <wsdl:message name="removeResponse">
6293   <wsdl:part name="removeResponse" element="removeResponse"/>
6294 </wsdl:message>

```

```

6295
6296 <wsdl:message name="batchRemove">
6297   <wsdl:part name="batchRemove" element="batchRemove"/>
6298 </wsdl:message>
6299 <wsdl:message name="batchRemoveResponse">
6300   <wsdl:part name="batchRemoveResponse" element="batchRemoveResponse"/>
6301 </wsdl:message>
6302
6303 <wsdl:message name="fail">
6304   <wsdl:part name="fail" element="fail"/>
6305 </wsdl:message>
6306 <wsdl:message name="failResponse">
6307   <wsdl:part name="failResponse" element="failResponse"/>
6308 </wsdl:message>
6309
6310 <wsdl:message name="batchFail">
6311   <wsdl:part name="batchFail" element="batchFail"/>
6312 </wsdl:message>
6313 <wsdl:message name="batchFailResponse">
6314   <wsdl:part name="batchFailResponse" element="batchFailResponse"/>
6315 </wsdl:message>
6316
6317 <wsdl:message name="setPriority">
6318   <wsdl:part name="setPriority" element="setPriority"/>
6319 </wsdl:message>
6320 <wsdl:message name="setPriorityResponse">
6321   <wsdl:part name="setPriorityResponse" element="setPriorityResponse"/>
6322 </wsdl:message>
6323
6324 <wsdl:message name="batchSetPriority">
6325   <wsdl:part name="batchSetPriority" element="batchSetPriority"/>
6326 </wsdl:message>
6327 <wsdl:message name="batchSetPriorityResponse">
6328   <wsdl:part name="batchSetPriorityResponse"
6329 element="batchSetPriorityResponse"/>
6330 </wsdl:message>
6331
6332 <wsdl:message name="addAttachment">
6333   <wsdl:part name="addAttachment" element="addAttachment"/>
6334 </wsdl:message>
6335 <wsdl:message name="addAttachmentResponse">
6336   <wsdl:part name="addAttachmentResponse" element="addAttachmentResponse"/>
6337 </wsdl:message>
6338
6339 <wsdl:message name="getAttachmentInfos">
6340   <wsdl:part name="getAttachmentInfos" element="getAttachmentInfos"/>
6341 </wsdl:message>
6342 <wsdl:message name="getAttachmentInfosResponse">
6343   <wsdl:part name="getAttachmentInfosResponse"
6344 element="getAttachmentInfosResponse"/>
6345 </wsdl:message>
6346
6347 <wsdl:message name="getAttachment">
6348   <wsdl:part name="getAttachment" element="getAttachment"/>
6349 </wsdl:message>
6350 <wsdl:message name="getAttachmentResponse">
6351   <wsdl:part name="getAttachmentResponse" element="getAttachmentResponse"/>
6352 </wsdl:message>

```

```
6353
6354     <wsdl:message name="deleteAttachment" >
6355         <wsdl:part name="deleteAttachment" element="deleteAttachment" />
6356     </wsdl:message>
6357     <wsdl:message name="deleteAttachmentResponse" >
6358         <wsdl:part name="deleteAttachmentResponse"
6359 element="deleteAttachmentResponse" />
6360     </wsdl:message>
6361
6362     <wsdl:message name="addComment" >
6363         <wsdl:part name="addComment" element="addComment" />
6364     </wsdl:message>
6365     <wsdl:message name="addCommentResponse" >
6366         <wsdl:part name="addCommentResponse" element="addCommentResponse" />
6367     </wsdl:message>
6368
6369     <wsdl:message name="getComments" >
6370         <wsdl:part name="getComments" element="getComments" />
6371     </wsdl:message>
6372     <wsdl:message name="getCommentsResponse" >
6373         <wsdl:part name="getCommentsResponse" element="getCommentsResponse" />
6374     </wsdl:message>
6375
6376     <wsdl:message name="skip" >
6377         <wsdl:part name="skip" element="skip" />
6378     </wsdl:message>
6379     <wsdl:message name="skipResponse" >
6380         <wsdl:part name="skipResponse" element="skipResponse" />
6381     </wsdl:message>
6382
6383     <wsdl:message name="batchSkip" >
6384         <wsdl:part name="batchSkip" element="batchSkip" />
6385     </wsdl:message>
6386     <wsdl:message name="batchSkipResponse" >
6387         <wsdl:part name="batchSkipResponse" element="batchSkipResponse" />
6388     </wsdl:message>
6389
6390     <wsdl:message name="forward" >
6391         <wsdl:part name="forward" element="forward" />
6392     </wsdl:message>
6393     <wsdl:message name="forwardResponse" >
6394         <wsdl:part name="forwardResponse" element="forwardResponse" />
6395     </wsdl:message>
6396
6397     <wsdl:message name="batchForward" >
6398         <wsdl:part name="batchForward" element="batchForward" />
6399     </wsdl:message>
6400     <wsdl:message name="batchForwardResponse" >
6401         <wsdl:part name="batchForwardResponse" element="batchForwardResponse" />
6402     </wsdl:message>
6403
6404     <wsdl:message name="delegate" >
6405         <wsdl:part name="delegate" element="delegate" />
6406     </wsdl:message>
6407     <wsdl:message name="delegateResponse" >
6408         <wsdl:part name="delegateResponse" element="delegateResponse" />
6409     </wsdl:message>
6410
```

```

6411 <wsdl:message name="batchDelegate">
6412   <wsdl:part name="batchDelegate" element="batchDelegate" />
6413 </wsdl:message>
6414 <wsdl:message name="batchDelegateResponse">
6415   <wsdl:part name="batchDelegateResponse" element="batchDelegateResponse" />
6416 </wsdl:message>
6417
6418 <wsdl:message name="getRendering">
6419   <wsdl:part name="getRendering" element="getRendering" />
6420 </wsdl:message>
6421 <wsdl:message name="getRenderingResponse">
6422   <wsdl:part name="getRenderingResponse" element="getRenderingResponse" />
6423 </wsdl:message>
6424
6425 <wsdl:message name="getRenderingTypes">
6426   <wsdl:part name="getRenderingTypes" element="getRenderingTypes" />
6427 </wsdl:message>
6428 <wsdl:message name="getRenderingTypesResponse">
6429   <wsdl:part name="getRenderingTypesResponse"
6430 element="getRenderingTypesResponse" />
6431 </wsdl:message>
6432
6433 <wsdl:message name="getTaskDetails">
6434   <wsdl:part name="getTaskDetails" element="getTaskDetails" />
6435 </wsdl:message>
6436 <wsdl:message name="getTaskDetailsResponse">
6437   <wsdl:part name="getTaskDetailsResponse"
6438 element="getTaskDetailsResponse" />
6439 </wsdl:message>
6440
6441 <wsdl:message name="getTaskDescription">
6442   <wsdl:part name="getTaskDescription" element="getTaskDescription" />
6443 </wsdl:message>
6444 <wsdl:message name="getTaskDescriptionResponse">
6445   <wsdl:part name="getTaskDescriptionResponse"
6446 element="getTaskDescriptionResponse" />
6447 </wsdl:message>
6448
6449 <wsdl:message name="setOutput">
6450   <wsdl:part name="setOutput" element="setOutput" />
6451 </wsdl:message>
6452 <wsdl:message name="setOutputResponse">
6453   <wsdl:part name="setOutputResponse" element="setOutputResponse" />
6454 </wsdl:message>
6455
6456 <wsdl:message name="deleteOutput">
6457   <wsdl:part name="deleteOutput" element="deleteOutput" />
6458 </wsdl:message>
6459 <wsdl:message name="deleteOutputResponse">
6460   <wsdl:part name="deleteOutputResponse" element="deleteOutputResponse" />
6461 </wsdl:message>
6462
6463 <wsdl:message name="setFault">
6464   <wsdl:part name="setFault" element="setFault" />
6465 </wsdl:message>
6466 <wsdl:message name="setFaultResponse">
6467   <wsdl:part name="setFaultResponse" element="setFaultResponse" />
6468 </wsdl:message>

```



```
6469
6470 <wsdl:message name="deleteFault">
6471   <wsdl:part name="deleteFault" element="deleteFault"/>
6472 </wsdl:message>
6473 <wsdl:message name="deleteFaultResponse">
6474   <wsdl:part name="deleteFaultResponse" element="deleteFaultResponse"/>
6475 </wsdl:message>
6476
6477 <wsdl:message name="getInput">
6478   <wsdl:part name="getInput" element="getInput"/>
6479 </wsdl:message>
6480 <wsdl:message name="getInputResponse">
6481   <wsdl:part name="getInputResponse" element="getInputResponse"/>
6482 </wsdl:message>
6483
6484 <wsdl:message name="getOutput">
6485   <wsdl:part name="getOutput" element="getOutput"/>
6486 </wsdl:message>
6487 <wsdl:message name="getOutputResponse">
6488   <wsdl:part name="getOutputResponse" element="getOutputResponse"/>
6489 </wsdl:message>
6490
6491 <wsdl:message name="getFault">
6492   <wsdl:part name="getFault" element="getFault"/>
6493 </wsdl:message>
6494 <wsdl:message name="getFaultResponse">
6495   <wsdl:part name="getFaultResponse" element="getFaultResponse"/>
6496 </wsdl:message>
6497
6498 <wsdl:message name="getMyTaskAbstracts">
6499   <wsdl:part name="getMyTaskAbstracts" element="getMyTaskAbstracts"/>
6500 </wsdl:message>
6501 <wsdl:message name="getMyTaskAbstractsResponse">
6502   <wsdl:part name="getMyTaskAbstractsResponse"
6503 element="getMyTaskAbstractsResponse"/>
6504 </wsdl:message>
6505
6506 <wsdl:message name="getMyTaskDetails">
6507   <wsdl:part name="getMyTaskDetails" element="getMyTaskDetails"/>
6508 </wsdl:message>
6509 <wsdl:message name="getMyTaskDetailsResponse">
6510   <wsdl:part name="getMyTaskDetailsResponse"
6511 element="getMyTaskDetailsResponse"/>
6512 </wsdl:message>
6513
6514 <wsdl:message name="query">
6515   <wsdl:part name="query" element="query"/>
6516 </wsdl:message>
6517 <wsdl:message name="queryResponse">
6518   <wsdl:part name="queryResponse" element="queryResponse"/>
6519 </wsdl:message>
6520
6521 <wsdl:message name="activate">
6522   <wsdl:part name="activate" element="activate"/>
6523 </wsdl:message>
6524 <wsdl:message name="activateResponse">
6525   <wsdl:part name="activateResponse" element="activateResponse"/>
6526 </wsdl:message>
```

```

6527
6528 <wsdl:message name="batchActivate">
6529   <wsdl:part name="batchActivate" element="batchActivate"/>
6530 </wsdl:message>
6531 <wsdl:message name="batchActivateResponse">
6532   <wsdl:part name="batchActivateResponse" element="batchActivateResponse"/>
6533 </wsdl:message>
6534
6535 <wsdl:message name="nominate">
6536   <wsdl:part name="nominate" element="nominate"/>
6537 </wsdl:message>
6538 <wsdl:message name="nominateResponse">
6539   <wsdl:part name="nominateResponse" element="nominateResponse"/>
6540 </wsdl:message>
6541
6542 <wsdl:message name="batchNominate">
6543   <wsdl:part name="batchNominate" element="batchNominate"/>
6544 </wsdl:message>
6545 <wsdl:message name="batchNominateResponse">
6546   <wsdl:part name="batchNominateResponse" element="batchNominateResponse"/>
6547 </wsdl:message>
6548
6549 <wsdl:message name="setGenericHumanRole">
6550   <wsdl:part name="setGenericHumanRole" element="setGenericHumanRole"/>
6551 </wsdl:message>
6552 <wsdl:message name="setGenericHumanRoleResponse">
6553   <wsdl:part name="setGenericHumanRoleResponse"
6554 element="setGenericHumanRoleResponse"/>
6555 </wsdl:message>
6556
6557 <wsdl:message name="batchSetGenericHumanRole">
6558   <wsdl:part name="batchSetGenericHumanRole"
6559 element="batchSetGenericHumanRole"/>
6560 </wsdl:message>
6561 <wsdl:message name="batchSetGenericHumanRoleResponse">
6562   <wsdl:part name="batchSetGenericHumanRoleResponse"
6563 element="batchSetGenericHumanRoleResponse"/>
6564 </wsdl:message>
6565
6566 <wsdl:message name="getOutcome">
6567   <wsdl:part name="getOutcome" element="getOutcome"/>
6568 </wsdl:message>
6569 <wsdl:message name="getOutcomeResponse">
6570   <wsdl:part name="getOutcomeResponse" element="getOutcomeResponse"/>
6571 </wsdl:message>
6572
6573 <wsdl:message name="getTaskOperations">
6574   <wsdl:part name="getTaskOperations" element="getTaskOperations"/>
6575 </wsdl:message>
6576 <wsdl:message name="getTaskOperationsResponse">
6577   <wsdl:part name="getTaskOperationsResponse"
6578 element="getTaskOperationsResponse"/>
6579 </wsdl:message>
6580
6581 <wsdl:message name="getTaskInstanceData">
6582   <wsdl:part name="getTaskInstanceData" element="getTaskInstanceData"/>
6583 </wsdl:message>
6584 <wsdl:message name="getTaskInstanceDataResponse">

```



```

6585     <wsdl:part name="getTaskInstanceDataResponse"
6586 element="getTaskInstanceDataResponse" />
6587   </wsdl:message>
6588
6589   <wsdl:message name="getTaskHistory">
6590     <wsdl:part name="getTaskHistory" element="getTaskHistory" />
6591   </wsdl:message>
6592   <wsdl:message name="getTaskHistoryResponse">
6593     <wsdl:part name="getTaskHistoryResponse"
6594 element="getTaskHistoryResponse" />
6595   </wsdl:message>
6596
6597   <wsdl:message name="setTaskStartDeadlineExpression">
6598     <wsdl:part name="setTaskStartDeadlineExpression"
6599 element="setTaskStartDeadlineExpression" />
6600   </wsdl:message>
6601   <wsdl:message name="setTaskStartDeadlineExpressionResponse">
6602     <wsdl:part name="setTaskStartDeadlineExpressionResponse"
6603 element="setTaskStartDeadlineExpressionResponse" />
6604   </wsdl:message>
6605
6606   <wsdl:message name="setTaskStartDurationExpression">
6607     <wsdl:part name="setTaskStartDurationExpression"
6608 element="setTaskStartDurationExpression" />
6609   </wsdl:message>
6610   <wsdl:message name="setTaskStartDurationExpressionResponse">
6611     <wsdl:part name="setTaskStartDurationExpressionResponse"
6612 element="setTaskStartDurationExpressionResponse" />
6613   </wsdl:message>
6614
6615   <wsdl:message name="setTaskCompletionDeadlineExpression">
6616     <wsdl:part name="setTaskCompletionDeadlineExpression"
6617 element="setTaskCompletionDeadlineExpression" />
6618   </wsdl:message>
6619   <wsdl:message name="setTaskCompletionDeadlineExpressionResponse">
6620     <wsdl:part name="setTaskCompletionDeadlineExpressionResponse"
6621 element="setTaskCompletionDeadlineExpressionResponse" />
6622   </wsdl:message>
6623
6624   <wsdl:message name="setTaskCompletionDurationExpression">
6625     <wsdl:part name="setTaskCompletionDurationExpression"
6626 element="setTaskCompletionDurationExpression" />
6627   </wsdl:message>
6628   <wsdl:message name="setTaskCompletionDurationExpressionResponse">
6629     <wsdl:part name="setTaskCompletionDurationExpressionResponse"
6630 element="setTaskCompletionDurationExpressionResponse" />
6631   </wsdl:message>
6632
6633   <!-- Declaration of fault messages -->
6634   <wsdl:message name="illegalStateFault">
6635     <wsdl:part name="illegalState" element="illegalState" />
6636   </wsdl:message>
6637   <wsdl:message name="illegalArgumentFault">
6638     <wsdl:part name="illegalArgument" element="illegalArgument" />
6639   </wsdl:message>
6640   <wsdl:message name="illegalAccessFault">
6641     <wsdl:part name="illegalAccess" element="illegalAccess" />
6642   </wsdl:message>

```

```

6643 <wsdl:message name="illegalOperationFault">
6644   <wsdl:part name="illegalOperation" element="illegalOperation"/>
6645 </wsdl:message>
6646 <wsdl:message name="recipientNotAllowed">
6647   <wsdl:part name="recipientNotAllowed" element="recipientNotAllowed"/>
6648 </wsdl:message>
6649
6650 <!-- Port type definition -->
6651 <wsdl:portType name="taskOperations">
6652
6653   <wsdl:operation name="claim">
6654     <wsdl:input message="claim"/>
6655     <wsdl:output message="claimResponse"/>
6656     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
6657     <wsdl:fault name="illegalArgumentFault"
6658 message="illegalArgumentFault"/>
6659     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
6660     <wsdl:fault name="illegalOperationFault"
6661 message="illegalOperationFault"/>
6662   </wsdl:operation>
6663
6664   <wsdl:operation name="batchClaim">
6665     <wsdl:input message="batchClaim"/>
6666     <wsdl:output message="batchClaimResponse"/>
6667   </wsdl:operation>
6668
6669   <wsdl:operation name="start">
6670     <wsdl:input message="start"/>
6671     <wsdl:output message="startResponse"/>
6672     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
6673     <wsdl:fault name="illegalArgumentFault"
6674 message="illegalArgumentFault"/>
6675     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
6676     <wsdl:fault name="illegalOperationFault"
6677 message="illegalOperationFault"/>
6678   </wsdl:operation>
6679
6680   <wsdl:operation name="batchStart">
6681     <wsdl:input message="batchStart"/>
6682     <wsdl:output message="batchStartResponse"/>
6683   </wsdl:operation>
6684
6685   <wsdl:operation name="stop">
6686     <wsdl:input message="stop"/>
6687     <wsdl:output message="stopResponse"/>
6688     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
6689     <wsdl:fault name="illegalArgumentFault"
6690 message="illegalArgumentFault"/>
6691     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
6692     <wsdl:fault name="illegalOperationFault"
6693 message="illegalOperationFault"/>
6694   </wsdl:operation>
6695
6696   <wsdl:operation name="batchStop">
6697     <wsdl:input message="batchStop"/>
6698     <wsdl:output message="batchStopResponse"/>
6699   </wsdl:operation>
6700

```

```

6701     <wsdl:operation name="release">
6702         <wsdl:input message="release" />
6703         <wsdl:output message="releaseResponse" />
6704         <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
6705         <wsdl:fault name="illegalArgumentFault"
6706 message="illegalArgumentFault" />
6707         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
6708         <wsdl:fault name="illegalOperationFault"
6709 message="illegalOperationFault" />
6710     </wsdl:operation>
6711
6712     <wsdl:operation name="batchRelease">
6713         <wsdl:input message="batchRelease" />
6714         <wsdl:output message="batchReleaseResponse" />
6715     </wsdl:operation>
6716
6717     <wsdl:operation name="suspend">
6718         <wsdl:input message="suspend" />
6719         <wsdl:output message="suspendResponse" />
6720         <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
6721         <wsdl:fault name="illegalArgumentFault"
6722 message="illegalArgumentFault" />
6723         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
6724         <wsdl:fault name="illegalOperationFault"
6725 message="illegalOperationFault" />
6726     </wsdl:operation>
6727
6728     <wsdl:operation name="batchSuspend">
6729         <wsdl:input message="batchSuspend" />
6730         <wsdl:output message="batchSuspendResponse" />
6731     </wsdl:operation>
6732
6733     <wsdl:operation name="suspendUntil">
6734         <wsdl:input message="suspendUntil" />
6735         <wsdl:output message="suspendUntilResponse" />
6736         <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
6737         <wsdl:fault name="illegalArgumentFault"
6738 message="illegalArgumentFault" />
6739         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
6740         <wsdl:fault name="illegalOperationFault"
6741 message="illegalOperationFault" />
6742     </wsdl:operation>
6743
6744     <wsdl:operation name="batchSuspendUntil">
6745         <wsdl:input message="batchSuspendUntil" />
6746         <wsdl:output message="batchSuspendUntilResponse" />
6747     </wsdl:operation>
6748
6749     <wsdl:operation name="resume">
6750         <wsdl:input message="resume" />
6751         <wsdl:output message="resumeResponse" />
6752         <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
6753         <wsdl:fault name="illegalArgumentFault"
6754 message="illegalArgumentFault" />
6755         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
6756         <wsdl:fault name="illegalOperationFault"
6757 message="illegalOperationFault" />
6758     </wsdl:operation>

```

```

6759
6760     <wsdl:operation name="batchResume">
6761         <wsdl:input message="batchResume" />
6762         <wsdl:output message="batchResumeResponse" />
6763     </wsdl:operation>
6764
6765     <wsdl:operation name="complete">
6766         <wsdl:input message="complete" />
6767         <wsdl:output message="completeResponse" />
6768         <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
6769         <wsdl:fault name="illegalArgumentFault"
6770 message="illegalArgumentFault" />
6771         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
6772         <wsdl:fault name="illegalOperationFault"
6773 message="illegalOperationFault" />
6774     </wsdl:operation>
6775
6776     <wsdl:operation name="batchComplete">
6777         <wsdl:input message="batchComplete" />
6778         <wsdl:output message="batchCompleteResponse" />
6779     </wsdl:operation>
6780
6781     <wsdl:operation name="remove">
6782         <wsdl:input message="remove" />
6783         <wsdl:output message="removeResponse" />
6784         <wsdl:fault name="illegalArgumentFault"
6785 message="illegalArgumentFault" />
6786         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
6787         <wsdl:fault name="illegalOperationFault"
6788 message="illegalOperationFault" />
6789     </wsdl:operation>
6790
6791     <wsdl:operation name="batchRemove">
6792         <wsdl:input message="batchRemove" />
6793         <wsdl:output message="batchRemoveResponse" />
6794     </wsdl:operation>
6795
6796     <wsdl:operation name="fail">
6797         <wsdl:input message="fail" />
6798         <wsdl:output message="failResponse" />
6799         <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
6800         <wsdl:fault name="illegalArgumentFault"
6801 message="illegalArgumentFault" />
6802         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
6803         <wsdl:fault name="illegalOperationFault"
6804 message="illegalOperationFault" />
6805     </wsdl:operation>
6806
6807     <wsdl:operation name="batchFail">
6808         <wsdl:input message="batchFail" />
6809         <wsdl:output message="batchFailResponse" />
6810     </wsdl:operation>
6811
6812     <wsdl:operation name="setPriority">
6813         <wsdl:input message="setPriority" />
6814         <wsdl:output message="setPriorityResponse" />
6815         <wsdl:fault name="illegalStateFault" message="illegalStateFault" />

```

```

6816     <wsdl:fault name="illegalArgumentFault"
6817 message="illegalArgumentFault" />
6818     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
6819     <wsdl:fault name="illegalOperationFault"
6820 message="illegalOperationFault" />
6821 </wsdl:operation>
6822
6823 <wsdl:operation name="batchSetPriority">
6824     <wsdl:input message="batchSetPriority" />
6825     <wsdl:output message="batchSetPriorityResponse" />
6826 </wsdl:operation>
6827
6828 <wsdl:operation name="addAttachment">
6829     <wsdl:input message="addAttachment" />
6830     <wsdl:output message="addAttachmentResponse" />
6831     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
6832     <wsdl:fault name="illegalArgumentFault"
6833 message="illegalArgumentFault" />
6834     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
6835     <wsdl:fault name="illegalOperationFault"
6836 message="illegalOperationFault" />
6837 </wsdl:operation>
6838
6839 <wsdl:operation name="getAttachmentInfos">
6840     <wsdl:input message="getAttachmentInfos" />
6841     <wsdl:output message="getAttachmentInfosResponse" />
6842     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
6843     <wsdl:fault name="illegalArgumentFault"
6844 message="illegalArgumentFault" />
6845     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
6846     <wsdl:fault name="illegalOperationFault"
6847 message="illegalOperationFault" />
6848 </wsdl:operation>
6849
6850 <wsdl:operation name="getAttachment">
6851     <wsdl:input message="getAttachment" />
6852     <wsdl:output message="getAttachmentResponse" />
6853     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
6854     <wsdl:fault name="illegalArgumentFault"
6855 message="illegalArgumentFault" />
6856     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
6857     <wsdl:fault name="illegalOperationFault"
6858 message="illegalOperationFault" />
6859 </wsdl:operation>
6860
6861 <wsdl:operation name="deleteAttachment">
6862     <wsdl:input message="deleteAttachment" />
6863     <wsdl:output message="deleteAttachmentResponse" />
6864     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
6865     <wsdl:fault name="illegalArgumentFault"
6866 message="illegalArgumentFault" />
6867     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
6868     <wsdl:fault name="illegalOperationFault"
6869 message="illegalOperationFault" />
6870 </wsdl:operation>
6871
6872 <wsdl:operation name="addComment">
6873     <wsdl:input message="addComment" />

```



```

6874     <wsdl:output message="addCommentResponse" />
6875     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
6876     <wsdl:fault name="illegalArgumentFault"
6877 message="illegalArgumentFault" />
6878     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
6879     <wsdl:fault name="illegalOperationFault"
6880 message="illegalOperationFault" />
6881     </wsdl:operation>
6882
6883     <wsdl:operation name="getComments">
6884     <wsdl:input message="getComments" />
6885     <wsdl:output message="getCommentsResponse" />
6886     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
6887     <wsdl:fault name="illegalArgumentFault"
6888 message="illegalArgumentFault" />
6889     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
6890     <wsdl:fault name="illegalOperationFault"
6891 message="illegalOperationFault" />
6892     </wsdl:operation>
6893
6894     <wsdl:operation name="skip">
6895     <wsdl:input message="skip" />
6896     <wsdl:output message="skipResponse" />
6897     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
6898     <wsdl:fault name="illegalArgumentFault"
6899 message="illegalArgumentFault" />
6900     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
6901     <wsdl:fault name="illegalOperationFault"
6902 message="illegalOperationFault" />
6903     </wsdl:operation>
6904
6905     <wsdl:operation name="batchSkip">
6906     <wsdl:input message="batchSkip" />
6907     <wsdl:output message="batchSkipResponse" />
6908     </wsdl:operation>
6909
6910     <wsdl:operation name="forward">
6911     <wsdl:input message="forward" />
6912     <wsdl:output message="forwardResponse" />
6913     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
6914     <wsdl:fault name="illegalArgumentFault"
6915 message="illegalArgumentFault" />
6916     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
6917     <wsdl:fault name="illegalOperationFault"
6918 message="illegalOperationFault" />
6919     </wsdl:operation>
6920
6921     <wsdl:operation name="batchForward">
6922     <wsdl:input message="batchForward" />
6923     <wsdl:output message="batchForwardResponse" />
6924     </wsdl:operation>
6925
6926     <wsdl:operation name="delegate">
6927     <wsdl:input message="delegate" />
6928     <wsdl:output message="delegateResponse" />
6929     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
6930     <wsdl:fault name="illegalArgumentFault"
6931 message="illegalArgumentFault" />

```

```

6932     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
6933     <wsdl:fault name="illegalOperationFault"
6934 message="illegalOperationFault" />
6935     <wsdl:fault name="recipientNotAllowed" message="recipientNotAllowed" />
6936 </wsdl:operation>
6937
6938 <wsdl:operation name="batchDelegate">
6939     <wsdl:input message="batchDelegate" />
6940     <wsdl:output message="batchDelegateResponse" />
6941 </wsdl:operation>
6942
6943 <wsdl:operation name="getRendering">
6944     <wsdl:input message="getRendering" />
6945     <wsdl:output message="getRenderingResponse" />
6946     <wsdl:fault name="illegalArgumentFault"
6947 message="illegalArgumentFault" />
6948 </wsdl:operation>
6949
6950 <wsdl:operation name="getRenderingTypes">
6951     <wsdl:input message="getRenderingTypes" />
6952     <wsdl:output message="getRenderingTypesResponse" />
6953     <wsdl:fault name="illegalArgumentFault"
6954 message="illegalArgumentFault" />
6955 </wsdl:operation>
6956
6957 <wsdl:operation name="getTaskDetails">
6958     <wsdl:input message="getTaskDetails" />
6959     <wsdl:output message="getTaskDetailsResponse" />
6960     <wsdl:fault name="illegalArgumentFault"
6961 message="illegalArgumentFault" />
6962 </wsdl:operation>
6963
6964 <wsdl:operation name="getTaskDescription">
6965     <wsdl:input message="getTaskDescription" />
6966     <wsdl:output message="getTaskDescriptionResponse" />
6967     <wsdl:fault name="illegalArgumentFault"
6968 message="illegalArgumentFault" />
6969 </wsdl:operation>
6970
6971 <wsdl:operation name="setOutput">
6972     <wsdl:input message="setOutput" />
6973     <wsdl:output message="setOutputResponse" />
6974     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
6975     <wsdl:fault name="illegalArgumentFault"
6976 message="illegalArgumentFault" />
6977     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
6978     <wsdl:fault name="illegalOperationFault"
6979 message="illegalOperationFault" />
6980 </wsdl:operation>
6981
6982 <wsdl:operation name="deleteOutput">
6983     <wsdl:input message="deleteOutput" />
6984     <wsdl:output message="deleteOutputResponse" />
6985     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
6986     <wsdl:fault name="illegalArgumentFault"
6987 message="illegalArgumentFault" />
6988     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />

```

```

6989     <wsdl:fault name="illegalOperationFault"
6990 message="illegalOperationFault" />
6991   </wsdl:operation>
6992
6993   <wsdl:operation name="setFault">
6994     <wsdl:input message="setFault" />
6995     <wsdl:output message="setFaultResponse" />
6996     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
6997     <wsdl:fault name="illegalArgumentFault"
6998 message="illegalArgumentFault" />
6999     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
7000     <wsdl:fault name="illegalOperationFault"
7001 message="illegalOperationFault" />
7002   </wsdl:operation>
7003
7004   <wsdl:operation name="deleteFault">
7005     <wsdl:input message="deleteFault" />
7006     <wsdl:output message="deleteFaultResponse" />
7007     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
7008     <wsdl:fault name="illegalArgumentFault"
7009 message="illegalArgumentFault" />
7010     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
7011     <wsdl:fault name="illegalOperationFault"
7012 message="illegalOperationFault" />
7013   </wsdl:operation>
7014
7015   <wsdl:operation name="getInput">
7016     <wsdl:input message="getInput" />
7017     <wsdl:output message="getInputResponse" />
7018     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
7019     <wsdl:fault name="illegalArgumentFault"
7020 message="illegalArgumentFault" />
7021     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
7022     <wsdl:fault name="illegalOperationFault"
7023 message="illegalOperationFault" />
7024   </wsdl:operation>
7025
7026   <wsdl:operation name="getOutput">
7027     <wsdl:input message="getOutput" />
7028     <wsdl:output message="getOutputResponse" />
7029     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
7030     <wsdl:fault name="illegalArgumentFault"
7031 message="illegalArgumentFault" />
7032     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
7033     <wsdl:fault name="illegalOperationFault"
7034 message="illegalOperationFault" />
7035   </wsdl:operation>
7036
7037   <wsdl:operation name="getFault">
7038     <wsdl:input message="getFault" />
7039     <wsdl:output message="getFaultResponse" />
7040     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
7041     <wsdl:fault name="illegalArgumentFault"
7042 message="illegalArgumentFault" />
7043     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
7044     <wsdl:fault name="illegalOperationFault"
7045 message="illegalOperationFault" />
7046   </wsdl:operation>

```



```

7047
7048     <wsdl:operation name="getMyTaskAbstracts">
7049         <wsdl:input message="getMyTaskAbstracts" />
7050         <wsdl:output message="getMyTaskAbstractsResponse" />
7051         <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
7052         <wsdl:fault name="illegalArgumentFault"
7053 message="illegalArgumentFault" />
7054         <wsdl:fault name="illegalOperationFault"
7055 message="illegalOperationFault" />
7056     </wsdl:operation>
7057
7058     <wsdl:operation name="getMyTaskDetails">
7059         <wsdl:input message="getMyTaskDetails" />
7060         <wsdl:output message="getMyTaskDetailsResponse" />
7061         <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
7062         <wsdl:fault name="illegalArgumentFault"
7063 message="illegalArgumentFault" />
7064         <wsdl:fault name="illegalOperationFault"
7065 message="illegalOperationFault" />
7066     </wsdl:operation>
7067
7068     <wsdl:operation name="query">
7069         <wsdl:input message="query" />
7070         <wsdl:output message="queryResponse" />
7071         <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
7072         <wsdl:fault name="illegalArgumentFault"
7073 message="illegalArgumentFault" />
7074     </wsdl:operation>
7075
7076     <wsdl:operation name="activate">
7077         <wsdl:input message="activate" />
7078         <wsdl:output message="activateResponse" />
7079         <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
7080         <wsdl:fault name="illegalArgumentFault"
7081 message="illegalArgumentFault" />
7082         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
7083         <wsdl:fault name="illegalOperationFault"
7084 message="illegalOperationFault" />
7085     </wsdl:operation>
7086
7087     <wsdl:operation name="batchActivate">
7088         <wsdl:input message="batchActivate" />
7089         <wsdl:output message="batchActivateResponse" />
7090     </wsdl:operation>
7091
7092     <wsdl:operation name="nominate">
7093         <wsdl:input message="nominate" />
7094         <wsdl:output message="nominateResponse" />
7095         <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
7096         <wsdl:fault name="illegalArgumentFault"
7097 message="illegalArgumentFault" />
7098         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
7099         <wsdl:fault name="illegalOperationFault"
7100 message="illegalOperationFault" />
7101     </wsdl:operation>
7102
7103     <wsdl:operation name="batchNominate">
7104         <wsdl:input message="batchNominate" />

```

```

7105     <wsdl:output message="batchNominateResponse" />
7106 </wsdl:operation>
7107
7108     <wsdl:operation name="setGenericHumanRole">
7109         <wsdl:input message="setGenericHumanRole" />
7110         <wsdl:output message="setGenericHumanRoleResponse" />
7111         <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
7112         <wsdl:fault name="illegalArgumentFault"
7113 message="illegalArgumentFault" />
7114         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
7115         <wsdl:fault name="illegalOperationFault"
7116 message="illegalOperationFault" />
7117     </wsdl:operation>
7118
7119     <wsdl:operation name="batchSetGenericHumanRole">
7120         <wsdl:input message="batchSetGenericHumanRole" />
7121         <wsdl:output message="batchSetGenericHumanRoleResponse" />
7122     </wsdl:operation>
7123
7124     <wsdl:operation name="getOutcome">
7125         <wsdl:input message="getOutcome" />
7126         <wsdl:output message="getOutcomeResponse" />
7127         <wsdl:fault name="illegalArgumentFault"
7128 message="illegalArgumentFault" />
7129         <wsdl:fault name="illegalOperationFault"
7130 message="illegalOperationFault" />
7131     </wsdl:operation>
7132
7133     <wsdl:operation name="getTaskOperations">
7134         <wsdl:input message="getTaskOperations" />
7135         <wsdl:output message="getTaskOperationsResponse" />
7136         <wsdl:fault name="illegalArgumentFault"
7137 message="illegalArgumentFault" />
7138         <wsdl:fault name="illegalOperationFault"
7139 message="illegalOperationFault" />
7140     </wsdl:operation>
7141
7142     <wsdl:operation name="getTaskInstanceData">
7143         <wsdl:input message="getTaskInstanceData" />
7144         <wsdl:output message="getTaskInstanceDataResponse" />
7145         <wsdl:fault name="illegalArgumentFault"
7146 message="illegalArgumentFault" />
7147         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
7148         <wsdl:fault name="illegalOperationFault"
7149 message="illegalOperationFault" />
7150     </wsdl:operation>
7151
7152     <wsdl:operation name="getTaskHistory">
7153         <wsdl:input message="getTaskHistory" />
7154         <wsdl:output message="getTaskHistoryResponse" />
7155         <wsdl:fault name="illegalArgumentFault"
7156 message="illegalArgumentFault" />
7157         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
7158         <wsdl:fault name="illegalOperationFault"
7159 message="illegalOperationFault" />
7160     </wsdl:operation>
7161
7162     <wsdl:operation name="setTaskStartDeadlineExpression">

```

```

7163     <wsdl:input message="setTaskStartDeadlineExpression" />
7164     <wsdl:output message="setTaskStartDeadlineExpressionResponse" />
7165     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
7166     <wsdl:fault name="illegalArgumentFault"
7167 message="illegalArgumentFault" />
7168     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
7169     <wsdl:fault name="illegalOperationFault"
7170 message="illegalOperationFault" />
7171     </wsdl:operation>
7172
7173     <wsdl:operation name="setTaskStartDurationExpression">
7174     <wsdl:input message="setTaskStartDurationExpression" />
7175     <wsdl:output message="setTaskStartDurationExpressionResponse" />
7176     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
7177     <wsdl:fault name="illegalArgumentFault"
7178 message="illegalArgumentFault" />
7179     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
7180     <wsdl:fault name="illegalOperationFault"
7181 message="illegalOperationFault" />
7182     </wsdl:operation>
7183
7184     <wsdl:operation name="setTaskCompletionDeadlineExpression">
7185     <wsdl:input message="setTaskCompletionDeadlineExpression" />
7186     <wsdl:output message="setTaskCompletionDeadlineExpressionResponse" />
7187     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
7188     <wsdl:fault name="illegalArgumentFault"
7189 message="illegalArgumentFault" />
7190     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
7191     <wsdl:fault name="illegalOperationFault"
7192 message="illegalOperationFault" />
7193     </wsdl:operation>
7194
7195     <wsdl:operation name="setTaskCompletionDurationExpression">
7196     <wsdl:input message="setTaskCompletionDurationExpression" />
7197     <wsdl:output message="setTaskCompletionDurationExpressionResponse" />
7198     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
7199     <wsdl:fault name="illegalArgumentFault"
7200 message="illegalArgumentFault" />
7201     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
7202     <wsdl:fault name="illegalOperationFault"
7203 message="illegalOperationFault" />
7204     </wsdl:operation>
7205
7206     </wsdl:portType>
7207 </wsdl:definitions>

```

7208

E. WS-HumanTask Parent API Port Type

```

7209 <?xml version="1.0" encoding="UTF-8"?>
7210 <!--
7211 Copyright (c) OASIS Open 2009. All Rights Reserved.
7212 -->
7213 <wsdl:definitions
7214   targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
7215 humantask/leantask/api/200803"
7216   xmlns="http://docs.oasis-open.org/ns/bpel4people/ws-
7217 humantask/leantask/api/200803"
7218   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
7219   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
7220   xmlns:htd="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803"
7221   xmlns:htt="http://docs.oasis-open.org/ns/bpel4people/ws-
7222 humantask/types/200803">
7223
7224   <wsdl:documentation>
7225     Web Service Definition for WS-HumanTask 1.1 - Operations for Task Parent
7226 Applications
7227   </wsdl:documentation>
7228
7229   <wsdl:types>
7230     <xsd:schema
7231       targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
7232 humantask/leantask/api/200803"
7233       elementFormDefault="qualified"
7234       blockDefault="#all">
7235
7236       <xsd:import
7237         namespace="http://docs.oasis-open.org/ns/bpel4people/ws-
7238 humantask/200803"
7239         schemaLocation="ws-humantask.xsd" />
7240       <xsd:import
7241         namespace="http://docs.oasis-open.org/ns/bpel4people/ws-
7242 humantask/types/200803"
7243         schemaLocation="ws-humantask-types.xsd" />
7244
7245       <!-- Input and output elements -->
7246       <xsd:element name="registerLeanTaskDefinition">
7247         <xsd:complexType>
7248           <xsd:sequence>
7249             <xsd:element name="taskDefinition" type="htd:tLeanTask" />
7250           </xsd:sequence>
7251         </xsd:complexType>
7252       </xsd:element>
7253       <xsd:element name="registerLeanTaskDefinitionResponse">
7254         <xsd:complexType>
7255           <xsd:sequence>
7256             <xsd:element name="taskName" type="xsd:NCName" />
7257           </xsd:sequence>
7258         </xsd:complexType>
7259       </xsd:element>
7260
7261       <xsd:element name="unregisterLeanTaskDefinition">

```

```

7262     <xsd:complexType>
7263         <xsd:sequence>
7264             <xsd:element name="taskName" type="xsd:NCName" />
7265         </xsd:sequence>
7266     </xsd:complexType>
7267 </xsd:element>
7268 <xsd:element name="unregisterLeanTaskDefinitionResponse">
7269     <xsd:complexType>
7270         <xsd:sequence>
7271             <xsd:element name="taskName" type="xsd:NCName" />
7272         </xsd:sequence>
7273     </xsd:complexType>
7274 </xsd:element>
7275
7276 <xsd:element name="listLeanTaskDefinitions">
7277     <xsd:complexType>
7278         <xsd:sequence>
7279             <xsd:annotation>
7280                 <xsd:documentation>Empty message</xsd:documentation>
7281             </xsd:annotation>
7282         </xsd:sequence>
7283     </xsd:complexType>
7284 </xsd:element>
7285 <xsd:element name="listLeanTaskDefinitionsResponse">
7286     <xsd:complexType>
7287         <xsd:sequence>
7288             <xsd:element name="leanTaskDefinitions">
7289                 <xsd:complexType>
7290                     <xsd:sequence>
7291                         <xsd:element name="leanTaskDefinition" type="htd:tLeanTask"
7292 minOccurs="0" maxOccurs="unbounded" />
7293                     </xsd:sequence>
7294                 </xsd:complexType>
7295             </xsd:element>
7296         </xsd:sequence>
7297     </xsd:complexType>
7298 </xsd:element>
7299
7300 <xsd:element name="createLeanTask">
7301     <xsd:complexType>
7302         <xsd:sequence>
7303             <xsd:element name="inputMessage">
7304                 <xsd:complexType>
7305                     <xsd:sequence>
7306                         <xsd:any processContents="lax" namespace="##any" />
7307                     </xsd:sequence>
7308                 </xsd:complexType>
7309             </xsd:element>
7310             <xsd:element name="taskDefinition" type="htd:tLeanTask"
7311 minOccurs="0"/>
7312                 <xsd:element name="taskName" type="xsd:NCName" minOccurs="0" />
7313             </xsd:sequence>
7314         </xsd:complexType>
7315     </xsd:element>
7316 <xsd:element name="createLeanTaskResponse">
7317     <xsd:complexType>
7318         <xsd:sequence>
7319             <xsd:element name="outputMessage">

```

```

7320         <xsd:complexType>
7321             <xsd:sequence>
7322                 <xsd:any processContents="lax" namespace="##any" />
7323             </xsd:sequence>
7324         </xsd:complexType>
7325     </xsd:element>
7326 </xsd:sequence>
7327 </xsd:complexType>
7328 </xsd:element>
7329
7330 <xsd:element name="createLeanTaskAsync">
7331     <xsd:complexType>
7332         <xsd:sequence>
7333             <xsd:element name="inputMessage">
7334                 <xsd:complexType>
7335                     <xsd:sequence>
7336                         <xsd:any processContents="lax" namespace="##any" />
7337                     </xsd:sequence>
7338                 </xsd:complexType>
7339             </xsd:element>
7340             <xsd:element name="taskDefinition" type="htd:tLeanTask"
7341 minOccurs="0"/>
7342                 <xsd:element name="taskName" type="xsd:NCName" minOccurs="0" />
7343             </xsd:sequence>
7344         </xsd:complexType>
7345     </xsd:element>
7346 <xsd:element name="createLeanTaskAsyncResponse">
7347     <xsd:complexType>
7348         <xsd:sequence>
7349             <xsd:annotation>
7350                 <xsd:documentation>Empty message</xsd:documentation>
7351             </xsd:annotation>
7352         </xsd:sequence>
7353     </xsd:complexType>
7354 </xsd:element>
7355
7356 <xsd:element name="createLeanTaskAsyncCallback">
7357     <xsd:complexType>
7358         <xsd:sequence>
7359             <xsd:element name="outputMessage">
7360                 <xsd:complexType>
7361                     <xsd:sequence>
7362                         <xsd:any processContents="lax" namespace="##any" />
7363                     </xsd:sequence>
7364                 </xsd:complexType>
7365             </xsd:element>
7366         </xsd:sequence>
7367     </xsd:complexType>
7368 </xsd:element>
7369
7370 <!-- Fault elements -->
7371 <xsd:element name="illegalState">
7372     <xsd:complexType>
7373         <xsd:sequence>
7374             <xsd:element name="status" type="htt:tStatus"/>
7375             <xsd:element name="message" type="xsd:string"/>
7376         </xsd:sequence>
7377     </xsd:complexType>

```



```

7378     </xsd:element>
7379
7380     <xsd:element name="illegalArgument" type="xsd:string"/>
7381
7382     <xsd:element name="illegalAccess" type="xsd:string"/>
7383
7384     </xsd:schema>
7385 </wsdl:types>
7386
7387 <!-- Declaration of messages -->
7388 <wsdl:message name="registerLeanTaskDefinition">
7389   <wsdl:part name="registerLeanTaskDefinition"
7390 element="registerLeanTaskDefinition"/>
7391 </wsdl:message>
7392 <wsdl:message name="registerLeanTaskDefinitionResponse">
7393   <wsdl:part name="registerLeanTaskDefinitionResponse"
7394 element="registerLeanTaskDefinitionResponse"/>
7395 </wsdl:message>
7396
7397 <wsdl:message name="unregisterLeanTaskDefinition">
7398   <wsdl:part name="unregisterLeanTaskDefinition"
7399 element="unregisterLeanTaskDefinition"/>
7400 </wsdl:message>
7401 <wsdl:message name="unregisterLeanTaskDefinitionResponse">
7402   <wsdl:part name="unregisterLeanTaskDefinitionResponse"
7403 element="unregisterLeanTaskDefinitionResponse"/>
7404 </wsdl:message>
7405
7406 <wsdl:message name="listLeanTaskDefinitions">
7407   <wsdl:part name="listLeanTaskDefinitions"
7408 element="listLeanTaskDefinitions"/>
7409 </wsdl:message>
7410 <wsdl:message name="listLeanTaskDefinitionsResponse">
7411   <wsdl:part name="listLeanTaskDefinitionsResponse"
7412 element="listLeanTaskDefinitionsResponse"/>
7413 </wsdl:message>
7414
7415 <wsdl:message name="createLeanTask">
7416   <wsdl:part name="createLeanTask" element="createLeanTask"/>
7417 </wsdl:message>
7418 <wsdl:message name="createLeanTaskResponse">
7419   <wsdl:part name="createLeanTaskResponse"
7420 element="createLeanTaskResponse"/>
7421 </wsdl:message>
7422
7423 <wsdl:message name="createLeanTaskAsync">
7424   <wsdl:part name="createLeanTaskAsync" element="createLeanTaskAsync"/>
7425 </wsdl:message>
7426 <wsdl:message name="createLeanTaskAsyncResponse">
7427   <wsdl:part name="createLeanTaskAsyncResponse"
7428 element="createLeanTaskAsyncResponse"/>
7429 </wsdl:message>
7430
7431 <wsdl:message name="createLeanTaskAsyncCallback">
7432   <wsdl:part name="createLeanTaskAsyncCallback"
7433 element="createLeanTaskAsyncCallback"/>
7434 </wsdl:message>
7435

```

```

7436 <!-- Declaration of fault messages -->
7437 <wsdl:message name="illegalStateFault">
7438   <wsdl:part name="illegalState" element="illegalState"/>
7439 </wsdl:message>
7440 <wsdl:message name="illegalArgumentFault">
7441   <wsdl:part name="illegalArgument" element="illegalArgument"/>
7442 </wsdl:message>
7443 <wsdl:message name="illegalAccessFault">
7444   <wsdl:part name="illegalAccess" element="illegalAccess"/>
7445 </wsdl:message>
7446
7447 <!-- Port type definitions -->
7448 <wsdl:portType name="leanTaskOperations">
7449
7450   <wsdl:operation name="registerLeanTaskDefinition">
7451     <wsdl:input message="registerLeanTaskDefinition"/>
7452     <wsdl:output message="registerLeanTaskDefinitionResponse"/>
7453     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7454     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7455   </wsdl:operation>
7456
7457   <wsdl:operation name="unregisterLeanTaskDefinition">
7458     <wsdl:input message="unregisterLeanTaskDefinition"/>
7459     <wsdl:output message="unregisterLeanTaskDefinitionResponse"/>
7460     <wsdl:fault name="illegalArgumentFault"
7461 message="illegalArgumentFault"/>
7462     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7463   </wsdl:operation>
7464
7465   <wsdl:operation name="listLeanTaskDefinitions">
7466     <wsdl:input message="listLeanTaskDefinitions"/>
7467     <wsdl:output message="listLeanTaskDefinitionsResponse"/>
7468     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7469   </wsdl:operation>
7470
7471   <wsdl:operation name="createLeanTask">
7472     <wsdl:input message="createLeanTask"/>
7473     <wsdl:output message="createLeanTaskResponse"/>
7474     <wsdl:fault name="illegalArgumentFault"
7475 message="illegalArgumentFault"/>
7476     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7477   </wsdl:operation>
7478
7479   <wsdl:operation name="createLeanTaskAsync">
7480     <wsdl:input message="createLeanTaskAsync"/>
7481     <wsdl:output message="createLeanTaskAsyncResponse"/>
7482     <wsdl:fault name="illegalArgumentFault"
7483 message="illegalArgumentFault"/>
7484     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7485   </wsdl:operation>
7486
7487 </wsdl:portType>
7488
7489 <wsdl:portType name="leanTaskCallbackOperations">
7490
7491   <wsdl:operation name="createLeanTaskAsyncCallback">
7492     <wsdl:input message="createLeanTaskAsyncCallback"/>
7493   </wsdl:operation>

```



```
7494
7495     </wsdl:portType>
7496
7497 </wsdl:definitions>
```

F. WS-HumanTask Protocol Handler Port Types

```

7499 <?xml version="1.0" encoding="UTF-8"?>
7500 <!--
7501 Copyright (c) OASIS Open 2009. All Rights Reserved.
7502 -->
7503 <wsdl:definitions
7504   targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
7505 humantask/protocol/200803"
7506   xmlns="http://docs.oasis-open.org/ns/bpel4people/ws-
7507 humantask/protocol/200803"
7508   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
7509   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
7510   xmlns:htp="http://docs.oasis-open.org/ns/bpel4people/ws-
7511 humantask/protocol/200803">
7512
7513   <wsdl:documentation>
7514     Web Service Definition for WS-HumanTask 1.1 - Operations WS-HumanTask
7515 Protocol Participants
7516   </wsdl:documentation>
7517
7518   <wsdl:types>
7519     <xsd:schema
7520       targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
7521 humantask/protocol/200803"
7522       elementFormDefault="qualified"
7523       blockDefault="#all">
7524
7525       <xsd:complexType name="tProtocolMsgType">
7526         <xsd:sequence>
7527           <xsd:any namespace="##other" processContents="lax" minOccurs="0"
7528             maxOccurs="unbounded" />
7529         </xsd:sequence>
7530         <xsd:anyAttribute namespace="##any" processContents="lax" />
7531       </xsd:complexType>
7532
7533       <xsd:element name="skipped" type="htp:tProtocolMsgType" />
7534       <xsd:element name="fault" type="htp:tProtocolMsgType" />
7535       <xsd:element name="exit" type="htp:tProtocolMsgType" />
7536
7537       <xsd:element name="responseAction" type="xsd:anyURI" />
7538       <xsd:element name="responseOperation" type="xsd:NCName" />
7539     </xsd:schema>
7540   </wsdl:types>
7541
7542   <wsdl:message name="skipped">
7543     <wsdl:part name="parameters" element="skipped" />
7544   </wsdl:message>
7545   <wsdl:message name="fault">
7546     <wsdl:part name="parameters" element="fault" />
7547   </wsdl:message>
7548   <wsdl:message name="exit">
7549     <wsdl:part name="parameters" element="exit" />
7550   </wsdl:message>
7551

```

```
7552
7553 <wsdl:portType name="clientParticipantPortType">
7554   <wsdl:operation name="skippedOperation">
7555     <wsdl:input message="skipped" />
7556   </wsdl:operation>
7557   <wsdl:operation name="faultOperation">
7558     <wsdl:input message="fault" />
7559   </wsdl:operation>
7560 </wsdl:portType>
7561
7562 <wsdl:portType name="humanTaskParticipantPortType">
7563   <wsdl:operation name="exitOperation">
7564     <wsdl:input message="exit" />
7565   </wsdl:operation>
7566 </wsdl:portType>
7567
7568 </wsdl:definitions>
```

7569

G. WS-HumanTask Context Schema

```
7570 <?xml version="1.0" encoding="UTF-8"?>
7571 <!--
7572 Copyright (c) OASIS Open 2009. All Rights Reserved.
7573 -->
7574 <xsd:schema
7575   targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
7576 humantask/context/200803"
7577   xmlns="http://docs.oasis-open.org/ns/bpel4people/ws-
7578 humantask/context/200803"
7579   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
7580   xmlns:htt="http://docs.oasis-open.org/ns/bpel4people/ws-
7581 humantask/types/200803"
7582   elementFormDefault="qualified"
7583   blockDefault="#all">
7584
7585   <xsd:annotation>
7586     <xsd:documentation>
7587       XML Schema for WS-HumanTask 1.1 - Human Task Context for Task
7588 Interactions
7589     </xsd:documentation>
7590   </xsd:annotation>
7591
7592   <!-- other namespaces -->
7593   <xsd:import
7594     namespace="http://www.w3.org/XML/1998/namespace"
7595     schemaLocation="http://www.w3.org/2001/xml.xsd" />
7596   <xsd:import
7597     namespace="http://docs.oasis-open.org/ns/bpel4people/ws-
7598 humantask/types/200803"
7599     schemaLocation="ws-humantask-types.xsd" />
7600
7601   <!-- human task context -->
7602   <xsd:element name="humanTaskRequestContext"
7603 type="tHumanTaskRequestContext" />
7604   <xsd:complexType name="tHumanTaskRequestContext">
7605     <xsd:complexContent>
7606       <xsd:extension base="tHumanTaskContextBase">
7607         <xsd:sequence>
7608           <xsd:element name="peopleAssignments" type="tPeopleAssignments"
7609 minOccurs="0" />
7610           <xsd:element name="isSkipable" type="xsd:boolean" minOccurs="0" />
7611           <xsd:element name="expirationTime" type="xsd:dateTime"
7612 minOccurs="0" />
7613           <xsd:element name="activationDeferralTime" type="xsd:dateTime"
7614 minOccurs="0" />
7615           <xsd:any namespace="##other" processContents="lax" minOccurs="0"
7616 maxOccurs="unbounded" />
7617         </xsd:sequence>
7618       </xsd:extension>
7619     </xsd:complexContent>
7620   </xsd:complexType>
7621   <xsd:element name="humanTaskResponseContext"
7622 type="tHumanTaskResponseContext" />
```

```

7623 <xsd:complexType name="tHumanTaskResponseContext">
7624 <xsd:complexContent>
7625 <xsd:extension base="tHumanTaskContextBase">
7626 <xsd:sequence>
7627 <xsd:element name="actualOwner" type="htt:tUser"/>
7628 <xsd:element name="actualPeopleAssignments"
7629 type="tPeopleAssignments"/>
7630 <xsd:element name="outcome" type="xsd:string" minOccurs="0"/>
7631 <xsd:any namespace="##other" processContents="lax" minOccurs="0"
7632 maxOccurs="unbounded"/>
7633 </xsd:sequence>
7634 </xsd:extension>
7635 </xsd:complexContent>
7636 </xsd:complexType>
7637 <xsd:complexType name="tHumanTaskContextBase" abstract="true">
7638 <xsd:sequence>
7639 <xsd:element name="priority" type="htt:tPriority" minOccurs="0"/>
7640 <xsd:element name="attachments" type="tAttachments" minOccurs="0"/>
7641 </xsd:sequence>
7642 </xsd:complexType>
7643
7644 <!-- people assignments -->
7645 <xsd:complexType name="tPeopleAssignments">
7646 <xsd:sequence>
7647 <xsd:element ref="genericHumanRole" minOccurs="0"
7648 maxOccurs="unbounded"/>
7649 </xsd:sequence>
7650 </xsd:complexType>
7651 <xsd:element name="genericHumanRole" type="tGenericHumanRole"
7652 abstract="true" block="restriction extension"/>
7653 <xsd:element name="potentialOwners" type="tGenericHumanRole"
7654 substitutionGroup="genericHumanRole"/>
7655 <xsd:element name="excludedOwners" type="tGenericHumanRole"
7656 substitutionGroup="genericHumanRole"/>
7657 <xsd:element name="taskInitiator" type="tGenericHumanRole"
7658 substitutionGroup="genericHumanRole"/>
7659 <xsd:element name="taskStakeholders" type="tGenericHumanRole"
7660 substitutionGroup="genericHumanRole"/>
7661 <xsd:element name="businessAdministrators" type="tGenericHumanRole"
7662 substitutionGroup="genericHumanRole"/>
7663 <xsd:element name="recipients" type="tGenericHumanRole"
7664 substitutionGroup="genericHumanRole"/>
7665 <xsd:complexType name="tGenericHumanRole">
7666 <xsd:sequence>
7667 <xsd:element ref="htt:organizationalEntity"/>
7668 </xsd:sequence>
7669 </xsd:complexType>
7670
7671 <!-- attachments -->
7672 <xsd:complexType name="tAttachments">
7673 <xsd:sequence>
7674 <xsd:element name="returnAttachments" type="tReturnAttachments"
7675 minOccurs="0"/>
7676 <xsd:element ref="htt:attachment" minOccurs="0" maxOccurs="unbounded"/>
7677 </xsd:sequence>
7678 </xsd:complexType>
7679 <xsd:simpleType name="tReturnAttachments">
7680 <xsd:restriction base="xsd:string">

```

```
7681     <xsd:enumeration value="all" />
7682     <xsd:enumeration value="newOnly" />
7683     <xsd:enumeration value="none" />
7684   </xsd:restriction>
7685 </xsd:simpleType>
7686
7687 </xsd:schema>
```

7688

H. WS-HumanTask Policy Assertion Schema

```
7689 <?xml version="1.0" encoding="UTF-8"?>
7690 <!--
7691 Copyright (c) OASIS Open 2009. All Rights Reserved.
7692 -->
7693 <xsd:schema
7694   targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
7695 humantask/policy/200803"
7696   xmlns="http://docs.oasis-open.org/ns/bpel4people/ws-
7697 humantask/policy/200803"
7698   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
7699   xmlns:wsp="http://www.w3.org/ns/ws-policy"
7700   elementFormDefault="qualified"
7701   blockDefault="#all">
7702
7703   <xsd:annotation>
7704     <xsd:documentation>
7705       XML Schema for WS-HumanTask 1.1 - WS-HumanTask Policy Assertion
7706     </xsd:documentation>
7707   </xsd:annotation>
7708
7709   <!-- other namespaces -->
7710   <xsd:import
7711     namespace="http://www.w3.org/ns/ws-policy"
7712     schemaLocation="http://www.w3.org/2007/02/ws-policy.xsd" />
7713
7714   <!-- ws-humantask policy assertion -->
7715   <xsd:element name="HumanTaskAssertion" type="tHumanTaskAssertion"/>
7716   <xsd:complexType name="tHumanTaskAssertion" >
7717     <xsd:attribute ref="wsp:Optional" />
7718     <xsd:anyAttribute namespace="##any" processContents="lax" />
7719   </xsd:complexType>
7720
7721 </xsd:schema>
```

7722

I. Sample

7723 This appendix contains the full sample used in this specification.

7724

WSDL Definition

```
7726 <?xml version="1.0" encoding="UTF-8"?>
7727 <!--
7728 Copyright (c) OASIS Open 2009. All Rights Reserved.
7729 -->
7730 <wsdl:definitions name="ClaimApproval"
7731 targetNamespace="http://www.example.com/claims"
7732 xmlns:tns="http://www.example.com/claims"
7733 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
7734 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
7735 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
7736
7737 <wsdl:documentation>
7738 Example for WS-HumanTask 1.1 - WS-HumanTask Task Interface Definition
7739 </wsdl:documentation>
7740
7741 <wsdl:types>
7742 <xsd:schema
7743 targetNamespace="http://www.example.com/claims"
7744 xmlns:tns="http://www.example.com/claims"
7745 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
7746 elementFormDefault="qualified">
7747 <xsd:element name="ClaimApprovalData">
7748 <xsd:complexType>
7749 <xsd:sequence>
7750 <xsd:element name="cust">
7751 <xsd:complexType>
7752 <xsd:sequence>
7753 <xsd:element name="id" type="xsd:string">
7754 </xsd:element>
7755 <xsd:element name="firstname" type="xsd:string">
7756 </xsd:element>
7757 <xsd:element name="lastname" type="xsd:string">
7758 </xsd:element>
7759 </xsd:sequence>
7760 </xsd:complexType>
7761 </xsd:element>
7762 <xsd:element name="amount" type="xsd:double" />
7763 <xsd:element name="region" type="xsd:string" />
7764 <xsd:element name="prio" type="xsd:int" />
7765 <xsd:element name="activateAt" type="xsd:dateTime" />
7766 </xsd:sequence>
7767 </xsd:complexType>
7768 </xsd:element>
7769 </xsd:schema>
7770 </wsdl:types>
7771
7772 <wsdl:message name="ClaimApprovalRequest">
7773 <wsdl:part name="ClaimApprovalRequest"
7774 element="tns:ClaimApprovalData" />
```



```

7775 </wsdl:message>
7776 <wsdl:message name="ClaimApprovalResponse">
7777   <wsdl:part name="ClaimApprovalResponse" type="xsd:boolean" />
7778 </wsdl:message>
7779 <wsdl:message name="notifyRequest">
7780   <wsdl:part name="firstname" type="xsd:string" />
7781   <wsdl:part name="lastname" type="xsd:string" />
7782 </wsdl:message>
7783
7784 <wsdl:portType name="ClaimsHandlingPT">
7785   <wsdl:operation name="approve">
7786     <wsdl:input message="tns:ClaimApprovalRequest" />
7787   </wsdl:operation>
7788   <wsdl:operation name="escalate">
7789     <wsdl:input message="tns:ClaimApprovalRequest" />
7790   </wsdl:operation>
7791 </wsdl:portType>
7792
7793 <wsdl:portType name="ClaimsHandlingCallbackPT">
7794   <wsdl:operation name="approvalResponse">
7795     <wsdl:input message="tns:ClaimApprovalResponse" />
7796   </wsdl:operation>
7797 </wsdl:portType>
7798
7799 <wsdl:portType name="ClaimApprovalReminderPT">
7800   <wsdl:operation name="notify">
7801     <wsdl:input message="tns:notifyRequest" />
7802   </wsdl:operation>
7803 </wsdl:portType>
7804
7805 </wsdl:definitions>

```

7806

7807 Human Interaction Definition

```

7808 <?xml version="1.0" encoding="UTF-8"?>
7809 <!--
7810 Copyright (c) OASIS Open 2009. All Rights Reserved.
7811 -->
7812 <htd:humanInteractions
7813   xmlns:htd="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803"
7814   xmlns:htt="http://docs.oasis-open.org/ns/bpel4people/ws-
7815 humantask/types/200803"
7816   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
7817   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
7818   xmlns:cl="http://www.example.com/claims/"
7819   xmlns:tns="http://www.example.com"
7820   targetNamespace="http://www.example.com"
7821   xsi:schemaLocation="http://docs.oasis-open.org/ns/bpel4people/ws-
7822 humantask/200803 ../../xml/ws-humantask.xsd">
7823
7824   <htd:documentation>
7825     Example for WS-HumanTask 1.1 - WS-HumanTask Task Definition
7826   </htd:documentation>
7827
7828   <htd:import importType="http://schemas.xmlsoap.org/wsdl/"
7829     location="ws-humantask-example-claim-approval.wsdl"
7830     namespace="http://www.example.com/claims/" />
7831

```

```

7832 <htd:logicalPeopleGroups>
7833
7834   <htd:logicalPeopleGroup name="regionalClerks">
7835     <htd:documentation xml:lang="en-US">
7836       The group of clerks responsible for the region specified.
7837     </htd:documentation>
7838     <htd:parameter name="region" type="xsd:string" />
7839   </htd:logicalPeopleGroup>
7840
7841   <htd:logicalPeopleGroup name="regionalManager">
7842     <htd:documentation xml:lang="en-US">
7843       The manager responsible for the region specified.
7844     </htd:documentation>
7845     <htd:parameter name="region" type="xsd:string" />
7846   </htd:logicalPeopleGroup>
7847
7848   <htd:logicalPeopleGroup name="clerksManager">
7849     <htd:documentation xml:lang="en-US">
7850       The manager of the clerk whose user ID is passed as parameter.
7851     </htd:documentation>
7852     <htd:parameter name="clerkUserID" type="xsd:string" />
7853   </htd:logicalPeopleGroup>
7854
7855   <htd:logicalPeopleGroup name="directorClaims">
7856     <htd:documentation xml:lang="en-US">
7857       The functional director responsible for claims processing.
7858     </htd:documentation>
7859   </htd:logicalPeopleGroup>
7860
7861 </htd:logicalPeopleGroups>
7862
7863 <htd:tasks>
7864
7865   <htd:task name="ApproveClaim">
7866     <htd:documentation xml:lang="en-US">
7867       This task is used to handle claims that require manual
7868       approval.
7869     </htd:documentation>
7870
7871     <htd:interface portType="cl:ClaimsHandlingPT"
7872       operation="approve"
7873       responsePortType="cl:ClaimsHandlingCallbackPT"
7874       responseOperation="approvalResponse" />
7875
7876     <htd:priority>
7877       htd:getInput("ClaimApprovalRequest")/prio
7878     </htd:priority>
7879
7880     <htd:peopleAssignments>
7881       <htd:potentialOwners>
7882         <htd:from logicalPeopleGroup="regionalClerks">
7883           <htd:argument name="region">
7884             htd:getInput("ClaimApprovalRequest")/region
7885           </htd:argument>
7886         </htd:from>
7887       </htd:potentialOwners>
7888
7889       <htd:businessAdministrators>

```

```

7890     <htd:from logicalPeopleGroup="regionalManager">
7891         <htd:argument name="region">
7892             htd:getInput("ClaimApprovalRequest")/region
7893         </htd:argument>
7894     </htd:from>
7895     </htd:businessAdministrators>
7896 </htd:peopleAssignments>
7897
7898 <htd:delegation potentialDelegates="nobody" />
7899
7900 <htd:presentationElements>
7901
7902     <htd:name xml:lang="en-US">Approve Claim</htd:name>
7903     <htd:name xml:lang="de-DE">
7904         Genehmigung der Schadensforderung
7905     </htd:name>
7906
7907     <htd:presentationParameters>
7908         <htd:presentationParameter name="firstname"
7909             type="xsd:string">
7910             htd:getInput("ClaimApprovalRequest")/cust/firstname
7911         </htd:presentationParameter>
7912         <htd:presentationParameter name="lastname"
7913             type="xsd:string">
7914             htd:getInput("ClaimApprovalRequest")/cust/lastname
7915         </htd:presentationParameter>
7916         <htd:presentationParameter name="euroAmount"
7917             type="xsd:double">
7918             htd:getInput("ClaimApprovalRequest")/amount
7919         </htd:presentationParameter>
7920     </htd:presentationParameters>
7921
7922     <htd:subject xml:lang="en-US">
7923         Approve the insurance claim for €$euroAmount$ on behalf of
7924         $firstname$ $lastname$
7925     </htd:subject>
7926     <htd:subject xml:lang="de-DE">
7927         Genehmigung der Schadensforderung über €$euroAmount$ für
7928         $firstname$ $lastname$
7929     </htd:subject>
7930
7931     <htd:description xml:lang="en-US" contentType="text/plain">
7932         Approve this claim following corporate guideline
7933         #4711.0815/7 ...
7934     </htd:description>
7935     <htd:description xml:lang="en-US" contentType="text/html">
7936         <![CDATA[
7937         <p>
7938             Approve this claim following corporate guideline
7939             <b>#4711.0815/7</b>
7940             ...
7941         </p>
7942         ]]>
7943     </htd:description>
7944     <htd:description xml:lang="de-DE" contentType="text/plain">
7945         Genehmigen Sie diese Schadensforderung entsprechend
7946         Richtlinie Nr. 4711.0815/7 ...
7947     </htd:description>

```

```

7948 <htd:description xml:lang="de-DE" contentType="text/html">
7949 <![CDATA[
7950 <p>
7951 Genehmigen Sie diese Schadensforderung entsprechend
7952 Richtlinie
7953 <b>Nr. 4711.0815/7</b>
7954 ...
7955 </p>
7956 ]]>
7957 </htd:description>
7958
7959 </htd:presentationElements>
7960
7961 <htd:deadlines>
7962
7963 <htd:startDeadline name="sendReminder">
7964 <htd:documentation xml:lang="en-US">
7965 If not started within 3 days, - escalation notifications
7966 are sent if the claimed amount is less than 10000 - to the
7967 task's potential owners to remind them or their todo - to
7968 the regional manager, if this approval is of high priority
7969 (0,1, or 2) - the task is reassigned to Alan if the
7970 claimed amount is greater than or equal 10000
7971 </htd:documentation>
7972 <htd:for>P3D</htd:for>
7973
7974 <htd:escalation name="reminder">
7975
7976 <htd:condition>
7977 <![CDATA[
7978 htd:getInput("ClaimApprovalRequest")/amount < 10000
7979 ]]>
7980 </htd:condition>
7981
7982 <htd:toParts>
7983 <htd:toPart name="firstname">
7984 htd:getInput("ClaimApprovalRequest","ApproveClaim")
7985 /firstname
7986 </htd:toPart>
7987 <htd:toPart name="lastname">
7988 htd:getInput("ClaimApprovalRequest","ApproveClaim")
7989 /lastname
7990 </htd:toPart>
7991 </htd:toParts>
7992 </htd:toParts>
7993
7994 <htd:localNotification
7995 reference="tns:ClaimApprovalReminder">
7996
7997 <htd:documentation xml:lang="en-US">
7998 Reuse the predefined notification
7999 "ClaimApprovalReminder". Overwrite the recipients with
8000 the task's potential owners.
8001 </htd:documentation>
8002
8003 <htd:peopleAssignments>
8004 <htd:recipients>
8005 <htd:from>

```

```

8006         htd:getPotentialOwners("ApproveClaim")
8007         </htd:from>
8008         </htd:recipients>
8009         </htd:peopleAssignments>
8010
8011     </htd:localNotification>
8012
8013 </htd:escalation>
8014
8015 <htd:escalation name="highPrio">
8016
8017     <htd:condition>
8018         <![CDATA[
8019             (htd:getInput("ClaimApprovalRequest")/amount < 10000
8020             && htd:getInput("ClaimApprovalRequest")/prio <= 2)
8021         ]]>
8022     </htd:condition>
8023
8024     <!-- task input implicitly passed to the notification -->
8025
8026     <htd:notification name="ClaimApprovalOverdue">
8027         <htd:documentation xml:lang="en-US">
8028             An inline defined notification using the approval data
8029             as its input.
8030         </htd:documentation>
8031
8032         <htd:interface portType="cl:ClaimsHandlingPT"
8033             operation="escalate" />
8034
8035         <htd:peopleAssignments>
8036             <htd:recipients>
8037                 <htd:from logicalPeopleGroup="regionalManager">
8038                     <htd:argument name="region">
8039                         htd:getInput("ClaimApprovalRequest")/region
8040                     </htd:argument>
8041                 </htd:from>
8042             </htd:recipients>
8043         </htd:peopleAssignments>
8044
8045         <htd:presentationElements>
8046             <htd:name xml:lang="en-US">
8047                 Claim approval overdue
8048             </htd:name>
8049             <htd:name xml:lang="de-DE">
8050                 Überfällige Schadensforderungsgenehmigung
8051             </htd:name>
8052         </htd:presentationElements>
8053
8054     </htd:notification>
8055
8056 </htd:escalation>
8057
8058 <htd:escalation name="highAmountReassign">
8059
8060     <htd:condition>
8061         <![CDATA[
8062             htd:getInput("ClaimApprovalRequest")/amount >= 10000
8063         ]]>

```

```

8064     </htd:condition>
8065
8066     <htd:reassignment>
8067         <htd:documentation>
8068             Reassign task to Alan if amount is greater than or
8069             equal 10000.
8070         </htd:documentation>
8071
8072         <htd:potentialOwners>
8073             <htd:from>
8074                 <htd:literal>
8075                     <htt:organizationalEntity>
8076                         <htt:user>Alan</htt:user>
8077                     </htt:organizationalEntity>
8078                 </htd:literal>
8079             </htd:from>
8080         </htd:potentialOwners>
8081
8082     </htd:reassignment>
8083
8084 </htd:escalation>
8085
8086 </htd:startDeadline>
8087
8088
8089 <htd:completionDeadline name="notifyManager">
8090     <htd:documentation xml:lang="en-US">
8091         When not completed within 3 hours after having been
8092         claimed, the manager of the clerk who claimed the activity
8093         is notified.
8094     </htd:documentation>
8095     <htd:for>PT3H</htd:for>
8096
8097     <htd:escalation name="delayedApproval">
8098
8099         <htd:notification name="ClaimApprovalOverdue">
8100             <htd:documentation xml:lang="en-US">
8101                 An inline defined notification using the approval data
8102                 as its input.
8103             </htd:documentation>
8104
8105             <htd:interface portType="cl:ClaimsHandlingPT"
8106                 operation="escalate" />
8107
8108             <htd:peopleAssignments>
8109                 <htd:recipients>
8110                     <htd:from logicalPeopleGroup="clerksManager">
8111                         <htd:argument name="clerkUserID">
8112                             htd:getActualOwner("ApproveClaim")
8113                         </htd:argument>
8114                     </htd:from>
8115                 </htd:recipients>
8116             </htd:peopleAssignments>
8117
8118             <htd:presentationElements>
8119                 <htd:name xml:lang="en-US">
8120                     Claim approval overdue
8121                 </htd:name>

```

```

8122         <htd:name xml:lang="de-DE" >
8123             Überfällige Schadensforderungsgenehmigung
8124         </htd:name>
8125     </htd:presentationElements>
8126
8127 </htd:notification>
8128
8129 </htd:escalation>
8130 </htd:completionDeadline>
8131
8132 <htd:completionDeadline name="notifyDirector">
8133     <htd:documentation xml:lang="en-US" >
8134         When not completed within 2 days after having been
8135         claimed, the functional director of claims processing is
8136         notified.
8137     </htd:documentation>
8138     <htd:for>P2D</htd:for>
8139
8140 <htd:escalation name="severelyDelayedApproval">
8141
8142     <htd:notification name="ClaimApprovalOverdue">
8143         <htd:documentation xml:lang="en-US" >
8144             An inline defined notification using the approval data
8145             as its input.
8146         </htd:documentation>
8147
8148         <htd:interface portType="cl:ClaimsHandlingPT"
8149             operation="escalate" />
8150
8151         <htd:peopleAssignments>
8152             <htd:recipients>
8153                 <htd:from logicalPeopleGroup="directorClaims">
8154                     <htd:argument name="clerkUserID">
8155                         htd:getActualOwner("ApproveClaim")
8156                     </htd:argument>
8157                 </htd:from>
8158             </htd:recipients>
8159         </htd:peopleAssignments>
8160
8161     <htd:presentationElements>
8162         <htd:name xml:lang="en-US" >
8163             Claim approval severely overdue
8164         </htd:name>
8165         <htd:name xml:lang="de-DE" >
8166             Hochgradig überfällige Schadensforderungsgenehmigung
8167         </htd:name>
8168     </htd:presentationElements>
8169
8170 </htd:notification>
8171
8172 </htd:escalation>
8173 </htd:completionDeadline>
8174
8175 </htd:deadlines>
8176
8177 </htd:task>
8178
8179 </htd:tasks>

```

```

8180
8181 <htd:notifications>
8182
8183 <htd:notification name="ClaimApprovalReminder">
8184 <htd:documentation xml:lang="en-US">
8185     This notification is used to remind people of pending
8186     out-dated claim approvals. Recipients of this notification
8187     maybe overridden when it is referenced.
8188 </htd:documentation>
8189
8190 <htd:interface portType="cl:ClaimApprovalReminderPT"
8191     operation="notify" />
8192
8193 <htd:peopleAssignments>
8194 <htd:recipients>
8195 <htd:from>
8196 <htd:literal>
8197 <htt:organizationalEntity>
8198 <htt:user>Alan</htt:user>
8199 <htt:user>Dieter</htt:user>
8200 <htt:user>Frank</htt:user>
8201 <htt:user>Gerhard</htt:user>
8202 <htt:user>Ivana</htt:user>
8203 <htt:user>Karsten</htt:user>
8204 <htt:user>Matthias</htt:user>
8205 <htt:user>Patrick</htt:user>
8206 </htt:organizationalEntity>
8207 </htd:literal>
8208 </htd:from>
8209 </htd:recipients>
8210 </htd:peopleAssignments>
8211
8212 <htd:presentationElements>
8213
8214 <htd:name xml:lang="en-US">Approve Claim</htd:name>
8215 <htd:name xml:lang="de-DE">
8216     Genehmigung der Schadensforderung
8217 </htd:name>
8218
8219 <htd:presentationParameters>
8220 <htd:presentationParameter name="firstname"
8221     type="xsd:string">
8222     htd:getInput("firstname")
8223 </htd:presentationParameter>
8224 <htd:presentationParameter name="lastname"
8225     type="xsd:string">
8226     htd:getInput("lastname")
8227 </htd:presentationParameter>
8228 <htd:presentationParameter name="id" type="xsd:string">
8229     htd:getInput("taskId")
8230 </htd:presentationParameter>
8231 </htd:presentationParameters>
8232
8233 <htd:subject xml:lang="en-US">
8234     Claim approval for $firstname$, $lastname$ is overdue. See
8235     task $id$.
8236 </htd:subject>
8237

```



```
8238     </htd:presentationElements>
8239
8240     </htd:notification>
8241
8242     </htd:notifications>
8243
8244 </htd:humanInteractions>
```

8245 J. Acknowledgements

8246 The following individuals have participated in the creation of this specification and are gratefully
8247 acknowledged:

8248

8249 **Members of the BPEL4People Technical Committee:**

8250 Phillip Allen, Microsoft Corporation

8251 Ashish Agrawal, Adobe Systems

8252 Mike Amend, BEA Systems, Inc.

8253 Stefan Baeuerle, SAP AG

8254 Charlton Barreto, Adobe Systems

8255 Justin Brunt, TIBCO Software Inc.

8256 Martin Chapman, Oracle Corporation

8257 Luc Clément, Active Endpoints, Inc.

8258 Manoj Das, Oracle Corporation

8259 Alireza Farhoush, TIBCO Software Inc.

8260 Mark Ford, Active Endpoints, Inc.

8261 Sabine Holz, SAP AG

8262 Dave Ings, IBM

8263 Gershon Janssen, Individual

8264 Diane Jordan, IBM

8265 Anish Karmarkar, Oracle Corporation

8266 Ulrich Keil, SAP AG

8267 Oliver Kieselbach, SAP AG

8268 Matthias Kloppmann, IBM

8269 Dieter König, IBM

8270 Marita Kruempelmann, SAP AG

8271 Frank Leymann, IBM

8272 Mark Little, Red Hat

8273 Alexander Malek, Microsoft Corporation

8274 Ashok Malhotra, Oracle Corporation

8275 Mike Marin, IBM

8276 Vinkesh Mehta, Deloitte Consulting LLP

8277 Jeff Mischkinsky, Oracle Corporation

8278 Ralf Mueller, Oracle Corporation

8279 Krasimir Nedkov, SAP AG

8280 Benjamin Notheis, SAP AG

8281 Michael Pellegrini, Active Endpoints, Inc.

8282 Hannah Petereit, SAP AG

8283 Gerhard Pfau, IBM

8284 Karsten Ploesser, SAP AG

8285 Ravi Rangaswamy, Oracle Corporation
8286 Alan Rickayzen, SAP AG
8287 Michael Rowley, BEA Systems, Inc.
8288 Ron Ten-Hove, Sun Microsystems
8289 Ivana Trickovic, SAP AG
8290 Alessandro Triglia, OSS Nokalva
8291 Claus von Riegen, SAP AG
8292 Peter Walker, Sun Microsystems
8293 Franz Weber, SAP AG
8294 Prasad Yendluri, Software AG, Inc.

8295

8296 **WS-HumanTask 1.0 Specification Contributors:**

8297 Ashish Agrawal, Adobe
8298 Mike Amend, BEA
8299 Manoj Das, Oracle
8300 Mark Ford, Active Endpoints
8301 Chris Keller, Active Endpoints
8302 Matthias Kloppmann, IBM
8303 Dieter König, IBM
8304 Frank Leymann, IBM
8305 Ralf Müller, Oracle
8306 Gerhard Pfau, IBM
8307 Karsten Plösser, SAP
8308 Ravi Rangaswamy, Oracle
8309 Alan Rickayzen, SAP
8310 Michael Rowley, BEA
8311 Patrick Schmidt, SAP
8312 Ivana Trickovic, SAP
8313 Alex Yiu, Oracle
8314 Matthias Zeller, Adobe

8315

8316 The following individuals have provided valuable input into the design of this specification: Dave Ings,
8317 Diane Jordan, Mohan Kamath, Ulrich Keil, Matthias Kruse, Kurt Lind, Jeff Mischkinsky, Bhagat Nainani,
8318 Michael Pellegrini, Lars Rueter, Frank Ryan, David Shaffer, Will Stallard, Cyrille Waguët, Franz Weber,
8319 and Eric Wittmann.

8321

L. Revision History

8322 [optional; should not be included in OASIS Standards]

8323

Revision	Date	Editor	Changes Made
WD-01	2008-03-12	Dieter König	First working draft created from submitted specification
WD-02	2008-03-13	Dieter König	Added specification editors Moved WSDL and XSD into separate artifacts
WD-02	2008-06-25	Ivana Trickovic	Resolution of Issue #4 incorporated into the document/section 2.4.2
WD-02	2008-06-25	Ivana Trickovic	Resolution of Issue #4 incorporated into the ws-humantask.xsd
WD-02	2008-06-25	Ivana Trickovic	Resolution of Issue #8 incorporated into the document/section 6.2
WD-02	2008-06-25	Ivana Trickovic	Resolution of Issue #9 incorporated into the document/section 4.6 (example), and ws-humantask "ClaimApproval" example and WSDL file
WD-02	2008-06-28	Dieter König	Resolution of Issue #13 applied to complete document and all separate XML artifacts
WD-02	2008-06-28	Dieter König	Resolution of Issue #21 applied to section 2
WD-02	2008-07-08	Ralf Mueller	Resolution of Issue #14 applied to section 6, ws-humantask-api.wsdl and ws-humantask-types.xsd
WD-02	2008-07-15	Luc Clément	Updated Section 6.2 specifying (xsd:nonNegativeInteger) as the type for priority
WD-02	2008-07-25	Krasimir Nedkov	Resolution of Issue #18 applied to this document and all related XML artifacts. Completed the resolution of Issue #7 by adding the attachmentType input parameter to the addAttachment operation in section 6.1.1.
WD-02	2008-07-29	Ralf Mueller	Update of resolution of issue #14 applied to section 3.4.4, 6.1.2 and ws-humantask-types.xsd
CD-01-rev-1	2008-09-24	Dieter König	Resolution of Issue #25 applied to section 3.4.3.1 and ws-humantask-types.xsd

Revision	Date	Editor	Changes Made
CD-01-rev-2	2008-10-02	Ralf Mueller	Resolution of Issue #17 applied to section 2.3 Resolution of Issue #24 applied to section 7 and ws-humantask-context.xsd
CD-01-rev-3	2008-10-20	Dieter König	Resolution of Issue #23 applied to section 3.2.1 Resolution of Issue #6 applied to section 6.2 Resolution of Issue #15 applied to section 6.2 Formatting (Word Document Map)
CD-01-rev-4	2008-10-29	Michael Rowley	Resolution of Issue #2 Resolution of Issue #40
CD-01-rev-5	2008-11-09	Vinkesh Mehta	Issue-12, Removed section 7.4.1, Modified XML artifacts in bpel4people.xsd, humantask.xsd, humantask-context.xsd
CD-01-rev-6	2008-11-10	Vinkesh Mehta	Issue-46, Section 6.1.1 wrap getFaultResponse values into single element
CD-01-rev-7	2008-11-10	Vinkesh Mehta	Issue-35, section 6.1.1 remove potential owners from the authorized list of suspended, suspendUntil and resume
CD-01-rev-8	2008-11-21	Ivana Trickovic	Issue-16, sections 1, 2, 3, and 6
CD-01-rev-9	2008-11-21	Dieter König	Issue-16, sections 4, 5
CD-01-rev10	2008-11-30	Vinkesh Mehta	Issue-16, sections 7,8,9,10,11 Appendix A through H
CD-01-rev11	2008-12-15	Vinkesh Mehta	Issue-16, Updates based upon Dieter's comments
CD-01-rev-12	2008-12-17	Ivana Trickovic	Issue-16, sections 1, 2, 3, and 6 updates based on comments
CD-01-rev-13	2008-12-17	Dieter König	Issue-16, sections 4, 5 updates based on comments
CD-01-rev-14	2008-12-23	Vinkesh Mehta	Issue-16, Updates based upon Ivana's comments
CD-01-rev-15	2009-01-06	Krasimir Nedkov	Issue-43. Added section 6.1.5, column "Authorization" removed from the tables in section 6.1, edited texts in section 6.1.
CD-02	2009-02-18	Luc Clément	Committee Draft 2
CD-02-rev-1	2009-02-20	Dieter König	Issue 20, sections 4, 4.7 and 6.1.1 Issue 50, sections 3, 4, 6, 7 (htd:→htt:) Issue 55, section 2.5.2 (import type xsd)

Revision	Date	Editor	Changes Made
			Issue 56, section 7.2 (tProtocolMsgType) Issue 60, section 6.1.1 (API fault type) Issue 61, sections 3.4.4, 6.1 (taskDetails)
CD-02-rev-2	2009-02-22	Luc Clément	Issue 68, section 8.2 (XML Infoset) – removal of erroneous statement regarding the source of the value for the responseOperation
CD-02-rev-3	2009-02-22	Michael Rowley	Issue 44, section 6.1.1 plus ws- humantask.xsd and ws-humantask- api.wsdl
CD-02-rev-4	2009-03-05	Dieter König	Action Item 17
CD-02-rev-5	2009-03-09	Ralf Mueller	Issue 70, section 6.1.2
CD-02-rev-6	2009-03-13	Dieter König	Issue 71, section 3.4 and 6.1
CD-02-rev-7	2009-03-18	Ivana Trickovic	Issue 77
CD-02-rev-8	2009-03-21	Luc Clément	Issue 78
CD-02-rev-9	2009-03-27	Ivana Trickovic	Issue 77 + minor editorial changes (footer)
CD-03	2009-04-15	Luc Clément	Committee Draft 3
CD-03-rev1	2009-04-15	Luc Clément	Issue 75
CD-03-rev2	2009-05-27	Michael Rowley	Issue 41, 36, 45
CD-03-rev3	2009-06-01	Ivana Trickovic	Issue 80, 42 (also ws-humantask- types.xsd updated)
CD-03-rev4	2009-06-01	Luc Clément	Issue 65 – Incorporation of an HT architecture section into Section 1
CD-03-rev5	2009-06-02	Michael Rowley	Issue 37, 38 and 39
CD-03-rev6	2009-06-03	Ivana Trickovic	Issue 63, 81 (also ws-humantask- context.xsd updated)
CD-04	2009-06-17	Luc Clément	Committee Draft 4
CD-04-rev1	2009-06-17	Luc Clément	Acknowledgement update
CD-04-rev2	2009-06-17	Luc Clément	Incorporate BP-79
CD-04-rev3	2009-06-25	Ivana Trickovic	Issue 73
CD-04-rev4	2009-06-29	Dieter König	Issue 69, 84, 85, 93, 96, 106 Consistency issues in API data types Text formatting in new sections
CD-04-rev5	2009-06-29	Ravi Rangaswamy	Issue 98, 99
CD-05-rev0	2009-07-15	Luc Clément	Committee Draft 5
CD-05-rev1	2009-07-15	Luc Clément	Issue 117

Revision	Date	Editor	Changes Made
CD-05-rev2	2009-07-18	Dieter König	Issue 100, 112, 115 Issue 79 revisited: task/leanTask schema
CD-05-rev3	2009-08-06	Dieter König	Issue 88, 101, 102, 113, 116, 119, 120, 121, 123, 124
CD-05-rev4	2009-08-08	Luc Clément	Issue 91, 92, 94, 95
CD-05-rev4	2009-08-12	Ravi Rangaswamy	Issue 97, 108
CD-05-rev5	2009-08-24	Ravi Rangaswamy	Issue 90, 118
CD-05-rev6	2009-09-02	Ivana Trickovic	Issue 83, 114; ws-humantask.xsd updated accordingly
CD-05-rev7	2009-09-09	Ralf Mueller	Issue 104
CD-05-rev8	2009-09-28	Dieter König	Issue 105, 109, 125
CD-05-rev9	2009-10-13	Ivana Trickovic	Issue 103, 111
CD-05-rev10	2009-10-22	Dieter König	Issue 82, 127, 128, 129 XML artifacts copied back to appendix
CD-05-rev11	2009-11-01	Luc Clément	Issues 130, 131, 132 OASIS Spec QA Checklist updates
CD-06-rev00	2009-11-01	Luc Clément	Committee Draft 6

8324