



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

Committee Draft 07 / Public Review Draft 02

03 March 2010

Specification URIs:

This Version:

<http://docs.oasis-open.org/bpel4people/ws-humantask-1.1-spec-cd-07.html>
<http://docs.oasis-open.org/bpel4people/ws-humantask-1.1-spec-cd-07.doc> (Authoritative format)
<http://docs.oasis-open.org/bpel4people/ws-humantask-1.1-spec-cd-07.pdf>

Previous 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>
<http://docs.oasis-open.org/bpel4people/ws-humantask-1.1-spec-cd-06.pdf>

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~~2010. 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", [insert specific trademarked names and abbreviations here] 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	10
2	Language Design	15
2.1	Dependencies on Other Specifications	15
2.1.1	Namespaces Referenced	15
2.2	Language Extensibility	15
2.3	Overall Language Structure	16
2.3.1	Syntax	16
2.3.2	Properties	16
2.4	Default use of XPath 1.0 as an Expression Language	18
3	Concepts	19
3.1	Generic Human Roles	19
3.2	Composite Tasks and Sub Tasks	20
3.2.1	Composite Tasks by Definition	20
3.2.2	Composite Tasks Created Adhoc at Runtime	20
3.3	Routing Patterns	20
3.4	Relationship of Composite Tasks and Routing Patterns	21
3.5	Assigning People	21
3.5.1	Using Logical People Groups	22
3.5.2	Using Literals	23
3.5.3	Using Expressions	24
3.5.4	Data Type for Organizational Entities	25
3.5.5	Subtasks	25
3.6	Task Rendering	26
3.7	Lean Tasks	26
3.8	Task Instance Data	27
3.8.1	Presentation Data	27
3.8.2	Context Data	27
3.8.3	Operational Data	27
3.8.4	Data Types for Task Instance Data	29
3.8.5	Sub Tasks	32
4	Human Tasks	34
4.1	Overall Syntax	34
4.2	Properties	35
4.3	Presentation Elements	36
4.4	Task Possible Outcomes	39

4.5	Elements for Rendering Tasks	39
4.6	Elements for Composite Tasks	40
4.7	Elements for People Assignment	41
4.7.1	Routing Patterns	42
4.8	Completion Behavior	44
4.8.1	Completion Conditions.....	45
4.8.2	Result Construction from Parallel Subtasks	46
4.9	Elements for Handling Timeouts and Escalations.....	50
4.10	Human Task Behavior and State Transitions.....	57
4.10.1	Normal processing of a Human Task	57
4.10.2	Releasing a Human Task	58
4.10.3	Delegating or Forwarding a Human Task.....	58
4.10.4	Sub Task Event Propagation	58
4.11	History of a Human Task.....	59
4.11.1	Task Event Types and Data	60
4.11.2	Retrieving the History	62
5	Lean Tasks	65
5.1	Overall Syntax	65
5.2	Properties	65
5.3	Message Schema.....	65
5.4	Example: ToDoTask.....	67
6	Notifications	68
6.1	Overall Syntax	68
6.2	Properties	69
6.3	Notification Behavior and State Transitions	69
7	Programming Interfaces.....	70
7.1	Operations for Client Applications	70
7.1.1	Participant Operations	70
7.1.2	Simple Query Operations	86
7.1.3	Advanced Query Operation	90
7.1.4	Administrative Operations.....	94
7.1.5	Operation Authorizations	95
7.2	XPath Extension Functions	98
8	Interoperable Protocol for Advanced Interaction with Human Tasks	108
8.1	Human Task Coordination Protocol Messages.....	110
8.2	Protocol Messages	111
8.2.1	Protocol Messages Received by a Task Parent.....	111
8.2.2	Protocol Messages Received by a Task	111
8.3	WSDL of the Protocol Endpoints.....	111
8.3.1	Protocol Endpoint of the Task Parent.....	111
8.3.2	Protocol Endpoint of the Task.....	112
8.4	Providing Human Task Context.....	112
8.4.1	SOAP Binding of Human Task Context.....	112

8.4.2	Overriding Task Definition People Assignments	113
8.5	Human Task Policy Assertion	114
9	Task Parent Interactions with Lean Tasks.....	115
9.1	Operations for Task Parent Applications.....	115
9.2	Lean Task Interactions	115
9.2.1	Register a Lean Task Definition.....	115
9.2.2	Unregister a Lean Task Definition	116
9.2.3	List Lean Task Definitions.....	116
9.2.4	Create a Lean Task	117
9.2.5	Endpoints for Lean Task Operations	118
10	Providing Callback Information for Human Tasks.....	120
10.1	EPR Information Model Extension	120
10.2	XML Infoset Representation.....	120
10.3	Message Addressing Properties	122
10.4	SOAP Binding.....	123
11	Security Considerations	126
12	Conformance.....	127
A.	Portability and Interoperability Considerations	128
B.	WS-HumanTask Language Schema	129
C.	WS-HumanTask Data Types Schema.....	144
D.	WS-HumanTask Client API Port Type.....	154
E.	WS-HumanTask Parent API Port Type.....	224
F.	WS-HumanTask Protocol Handler Port Types	230
G.	WS-HumanTask Context Schema	232
H.	WS-HumanTask Policy Assertion Schema.....	235
I.	Sample.....	236
J.	Acknowledgements.....	246
K.	Non-Normative Text.....	248
L.	Revision History	249

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

90

91 **[XML Schema Part 1]**

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

94 **[XML Schema Part 2]**

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

97 **[XMLSpec]**

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

100 **[XPath 1.0]**

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

103 **1.3 Non-Normative References**

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

105 **1.4 Conformance Targets**

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

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

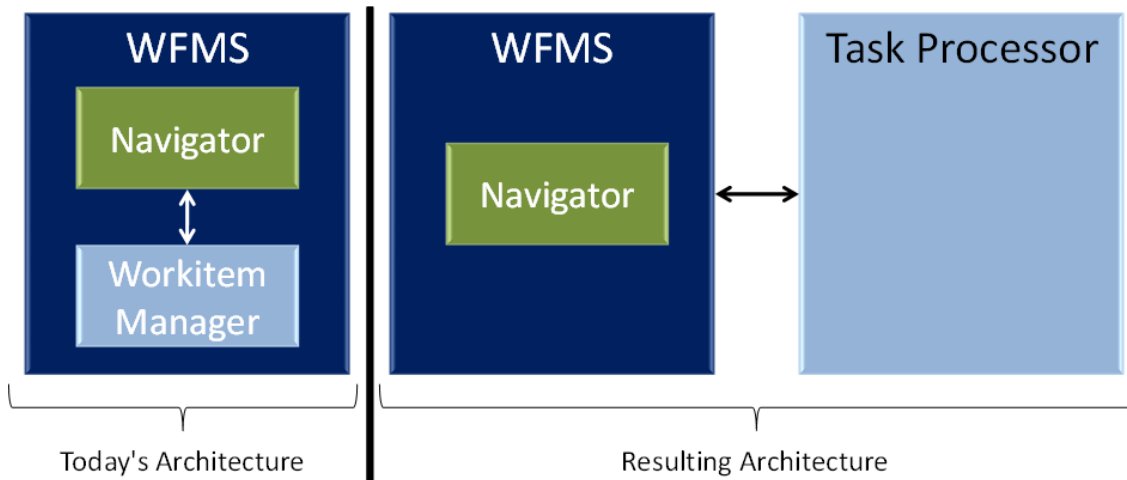
119

120

121 1.5 Overall Architecture

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

129



130

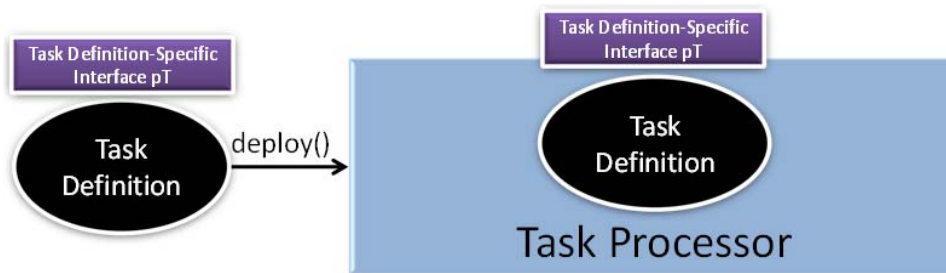
131 **Figure 1- Architectural Impact of WS-HumanTask on Workflow Management Systems**

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

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

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

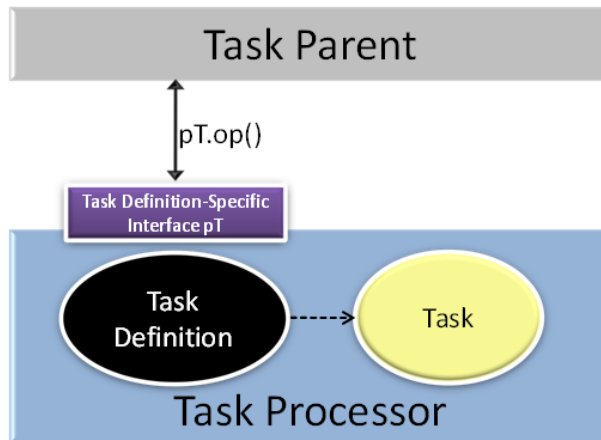
149



150
151
152
153
154
155
156
157

Figure 2 - Task Definitions Deployed in Task Processor

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



158
159

Figure 3 - Instantiating Tasks

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.

160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178

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.

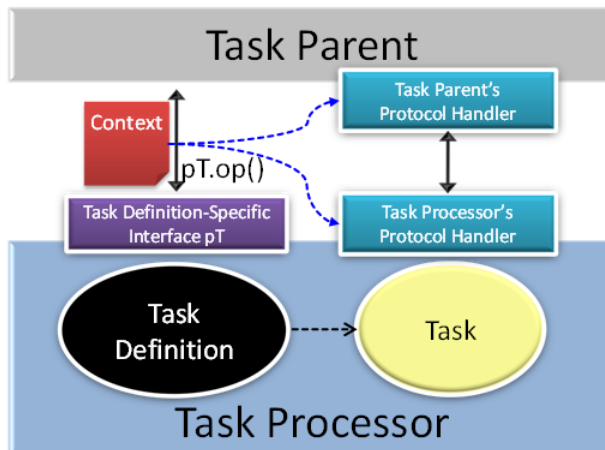


Figure 4 - Establishing a Protocol Channel

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

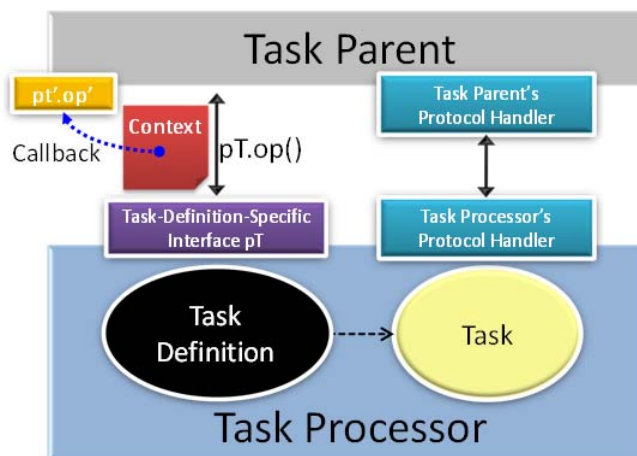


Figure 5 - Passing Callback Information for Long Running Tasks

189
 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.

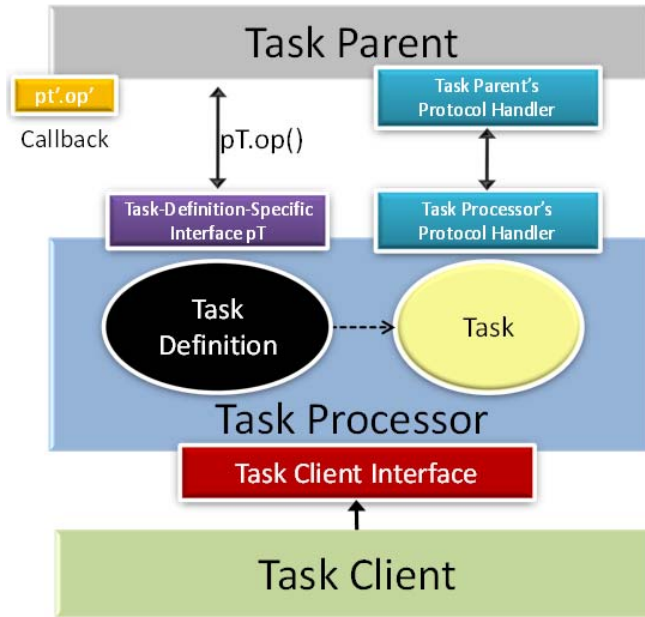
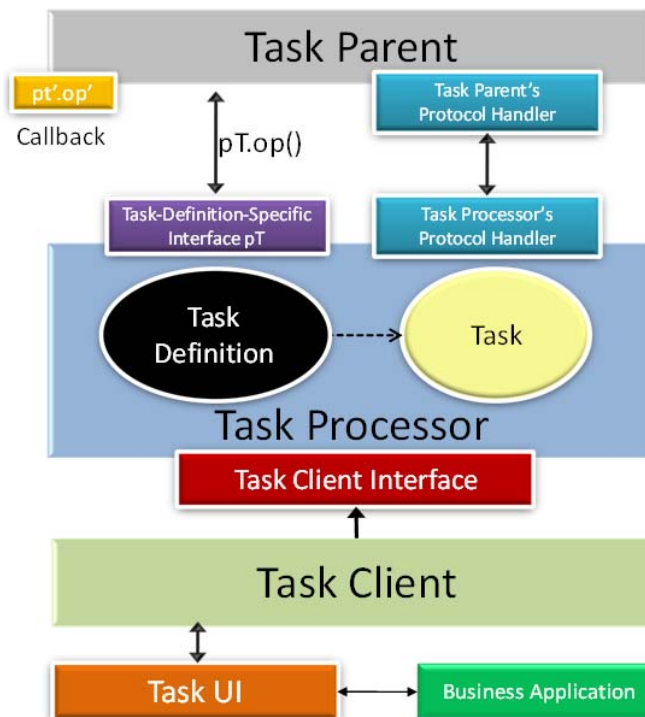


Figure 6 - Task List Client and Corresponding Interface

204
205
206
207
208
209
210
211
212
213
214
215

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

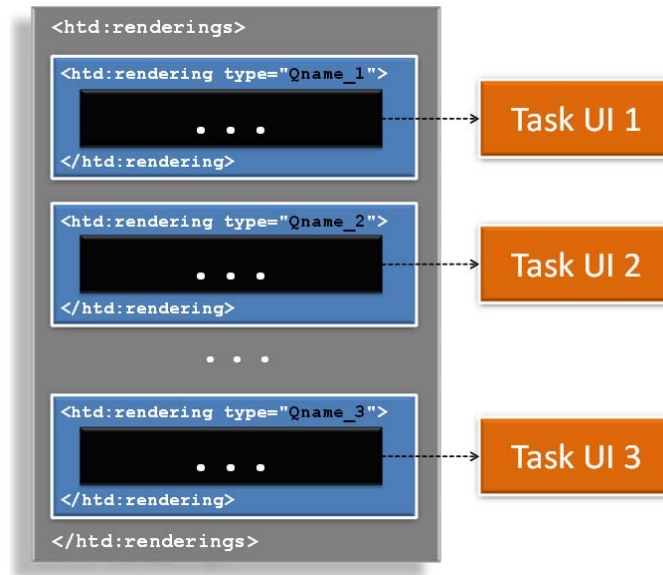


216

217

Figure 7 - Overall Architecture of a Human Task Infrastructure

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

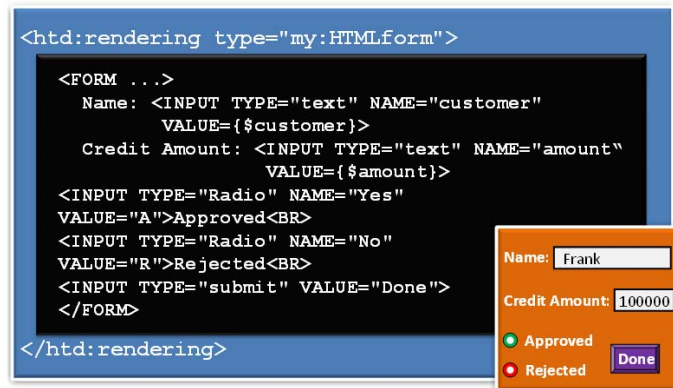


224

225

Figure 8 - Potential Renderings of a Task

226 For example Figure 9 depicts a rendering of type my:HTMLform. Its QName denotes that HTML forms
227 processing capabilities is needed to render the corresponding user interface of the task enclosing this
228 rendering. The nested element of the my:HTMLform rendering contains the actual HTML form to be
229 rendered. The example further assumes that the forms processor understands the {\$...} notation (see
230 section 4.3) to provide values from the task input as data presented in the form.



231

232

Figure 9 - Sample Rendering of a Task

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

236 2 Language Design

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

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

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

249 2.1 Dependencies on Other Specifications

250 WS-HumanTask utilizes the following specifications:

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

257 2.1.1 Namespaces Referenced

258 WS-HumanTask references these namespaces:

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

263 2.2 Language Extensibility

264 The WS-HumanTask extensibility mechanism allows:

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

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

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

274 2.3 Overall Language Structure

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

278 2.3.1 Syntax

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

313 2.3.2 Properties

314 The <humanInteractions> element has the following properties:

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

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

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

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

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

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

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

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

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

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

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

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

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

403 2.4 Default use of XPath 1.0 as an Expression Language

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

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

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

418

3 Concepts

419

3.1 Generic Human Roles

420

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:

421

422

- Task initiator

423

- Task stakeholders

424

- Potential owners

425

- Actual owner

426

- Excluded owners

427

- Business administrators

428

- Notification recipients

429

430

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.

431

432

433

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.

434

435

436

437

438

439

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.

440

441

442

443

444

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.

445

446

447

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.

448

449

450

451

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.

452

453

454

455

456

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.

457

458

459

460

461 **3.2 Composite Tasks and Sub Tasks**

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

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

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

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

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

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

476 **3.2.1 Composite Tasks by Definition**

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

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

492 **3.2.2 Composite Tasks Created Adhoc at Runtime**

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

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

498 **3.3 Routing Patterns**

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

504 3.4 Relationship of Composite Tasks and Routing Patterns

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

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

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

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

517 3.5 Assigning People

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

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

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

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

530 Syntax:

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

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

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

```
551
552 <htd:taskStakeholders>
553   <htd:from>...</htd:from>
554 </htd:taskStakeholders>
555
556 <htd:businessAdministrators>
557   <htd:from>...</htd:from>
558 </htd:businessAdministrators>
559
560 <htd:recipients>
561   <htd:from>...</htd:from>
562 </htd:recipients>
```

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

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

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

581 3.5.1 Using Logical People Groups

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

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

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

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

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

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

621 **Syntax:**

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

627
628 The `logicalPeopleGroup` attribute refers to a `logicalPeopleGroup` definition. The element
629 `<argument>` is used to pass values used in the people query. The `expressionLanguage` attribute
630 specifies the language used in the expression. The attribute is optional. If not specified, the default
631 language as inherited from the closest enclosing element that specifies the attribute MUST be used by
632 WS-HumanTask Processor.

633 **Example:**

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

641 **3.5.2 Using Literals**

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

645

646

647 **Syntax:**

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

653 **Example specifying user identifiers:**

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

670 **Example specifying group names:**

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

680 **3.5.3 Using Expressions**

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

684 **Syntax:**

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

688

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

692

693

694 **Example:**

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

705 **3.5.4 Data Type for Organizational Entities**

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

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

727 **3.5.5 Subtasks**

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

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

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

747 3.6 Task Rendering

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

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

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

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

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

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

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

777 3.7 Lean Tasks

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

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

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

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

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

804 **3.8 Task Instance Data**

805 Task instance data falls into three categories:

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

812 **3.8.1 Presentation Data**

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

816 **3.8.2 Context Data**

817 The task context includes the following:

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

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

827 **3.8.3 Operational Data**

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

831 3.8.3.1 Ad-hoc Attachments

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

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

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

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

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

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

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

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

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

858 Attachment Info Data Type

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

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

874 Attachment Data Type

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

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

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

883 3.8.3.2 Comments

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

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

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

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

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

895 Comment Data Type

896 The following data type is used to return comments.

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

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

911 3.8.4 Data Types for Task Instance Data

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

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

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

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

925 TaskAbstract Data Type

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

```

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

```

979 TaskDetails Data Type

```

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

```



```

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

```

```

1042         type="xsd:string" minOccurs="0"/>
1043     <xsd:element name="outcome"
1044         type="xsd:string" minOccurs="0"/>
1045     <xsd:element name="parentTaskId"
1046         type="xsd:stringanyURI" minOccurs="0"/>
1047     <xsd:element name="hasSubTasks"
1048         type="xsd:boolean" minOccurs="0"/>
1049     <xsd:any namespace="##other" processContents="lax"
1050         minOccurs="0" maxOccurs="unbounded"/>
1051 </xsd:sequence>
1052 </xsd:complexType>

```

Common Data Types

```

1053 <xsd:simpleType name="tPresentationName">
1054     <xsd:annotation>
1055         <xsd:documentation>length-restricted string</xsd:documentation>
1056     </xsd:annotation>
1057     <xsd:restriction base="xsd:string">
1058         <xsd:maxLength value="64" />
1059         <xsd:whiteSpace value="preserve" />
1060     </xsd:restriction>
1061 </xsd:simpleType>
1062
1063 <xsd:simpleType name="tPresentationSubject">
1064     <xsd:annotation>
1065         <xsd:documentation>length-restricted string</xsd:documentation>
1066     </xsd:annotation>
1067     <xsd:restriction base="xsd:string">
1068         <xsd:maxLength value="254" />
1069         <xsd:whiteSpace value="preserve" />
1070     </xsd:restriction>
1071 </xsd:simpleType>
1072
1073 <xsd:simpleType name="tStatus">
1074     <xsd:restriction base="xsd:string" />
1075 </xsd:simpleType>
1076
1077 <xsd:simpleType name="tPredefinedStatus">
1078     <xsd:annotation>
1079         <xsd:documentation>for documentation only</xsd:documentation>
1080     </xsd:annotation>
1081     <xsd:restriction base="xsd:string">
1082         <xsd:enumeration value="CREATED" />
1083         <xsd:enumeration value="READY" />
1084         <xsd:enumeration value="RESERVED" />
1085         <xsd:enumeration value="IN_PROGRESS" />
1086         <xsd:enumeration value="SUSPENDED" />
1087         <xsd:enumeration value="COMPLETED" />
1088         <xsd:enumeration value="FAILED" />
1089         <xsd:enumeration value="ERROR" />
1090         <xsd:enumeration value="EXITED" />
1091         <xsd:enumeration value="OBSOLETE" />
1092     </xsd:restriction>
1093 </xsd:simpleType>
1094

```

3.8.5 Sub Tasks

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

- 1097 • sub tasks → A list of task identifiers for each already-created subtask of the task, including
- 1098 both non-terminated and terminated instances
- 1099 → A list of the names of the sub tasks available for creation in the definition of the
- 1100 task, based on the composition type, instantiation pattern, and already created tasks
- 1101 • parent task → The identifier of the superior composite task of this task if it is a sub task

1102

4 Human Tasks

1103 The <task> element is used to specify human tasks. This section introduces the syntax for the element,
1104 and individual properties are explained in subsequent sections.

1105

4.1 Overall Syntax

1106

Definition of human tasks:

1107

```
<htd:task name="NCName" actualOwnerRequired="yes|no"?>
```

1108

```
  <htd:interface portType="QName" operation="NCName"  
    responsePortType="QName"? responseOperation="NCName"? />
```

1109

1110

```
  <htd:priority expressionLanguage="anyURI"? >?
```

1111

```
    integer-expression
```

1112

```
  </htd:priority>
```

1113

1114

```
  <htd:peopleAssignments>?
```

1115

```
    ...
```

1116

```
  </htd:peopleAssignments>
```

1117

1118

```
  <htd:completionBehavior>?
```

1119

```
    ...
```

1120

```
  </htd:completionBehavior>
```

1121

1122

```
  <htd:delegation  
    potentialDelegates="anybody|nobody|potentialOwners|other" />?
```

1123

```
    <htd:from>?
```

1124

```
      ...
```

1125

```
    </htd:from>
```

1126

```
  </htd:delegation>
```

1127

1128

```
  <htd:presentationElements>?
```

1129

```
    ...
```

1130

```
  </htd:presentationElements>
```

1131

1132

```
  <htd:possibleOutcomes>?
```

1133

```
    ...
```

1134

```
  </htd:possibleOutcomes>
```

1135

1136

```
  <htd:outcome part="NCName" queryLanguage="anyURI">?
```

1137

```
    queryContent
```

1138

```
  </htd:outcome>
```

1139

1140

```
  <htd:searchBy expressionLanguage="anyURI"? >?
```

1141

```
    expression
```

1142

```
  </htd:searchBy>
```

1143

1144

```
  <htd:renderings>?
```

1145

```
    <htd:rendering type="QName">+
```

1146

```
      ...
```

1147

```
    </htd:rendering>
```

1148

```
  </htd:renderings>
```

1149

```
  <htd:deadlines>?
```

1150

```

1154 <htd:startDeadline name="NCName">*
1155   ...
1156 </htd:startDeadline>
1157
1158 <htd:completionDeadline name="NCName">*
1159   ...
1160 </htd:completionDeadline>
1161
1162 </htd:deadlines>
1163
1164 <htd:composition>?
1165   ...
1166 </htd:composition>
1167
1168 </htd:task>
1169

```

1170 4.2 Properties

1171 The following attributes and elements are defined for tasks:

- 1172 • `name`: This attribute is used to specify the name of the task. The name combined with the target
1173 namespace MUST uniquely identify a task element enclosed in the task definition. This attribute
1174 is mandatory. It is not used for task rendering.
- 1175 • `actualOwnerRequired`: This optional attribute specifies if an actual owner is required for the
1176 task. Setting the value to "no" is used for composite tasks where subtasks should be activated
1177 automatically without user interaction. For routing tasks this attribute MUST be set to "no". Tasks
1178 that have been defined to not have subtasks MUST have exactly one actual owner after they
1179 have been claimed. For these tasks the value of the attribute value MUST be "yes". The default
1180 value for the attribute is "yes".
- 1181 • `interface`: This element is used to specify the operation used to invoke the task. The operation
1182 is specified using WSDL, that is, a WSDL port type and WSDL operation are defined. The
1183 element and its `portType` and `operation` attributes MUST be present for normal tasks. The
1184 schema only marks it optional so that Lean Tasks can make it prohibited. The interface is
1185 specified in one of the following forms:
 - 1186 ▪ The WSDL operation is a **one-way** operation and the task asynchronously
1187 returns output data. In this case, a WS-HumanTask Definition MUST specify a
1188 callback one-way operation, using the `responsePortType` and
1189 `responseOperation` attributes. This callback operation is invoked when the
1190 task has finished. The Web service endpoint address of the callback operation is
1191 provided at runtime when the task's one-way operation is invoked (for details,
1192 see section 10 "Providing Callback Information for Human Tasks").
 - 1193 ▪ The WSDL operation is a **request-response** operation. In this case, the
1194 `responsePortType` and `responseOperation` attributes MUST NOT be
1195 specified.
- 1196 • `priority`: This element is used to specify the priority of the task. It is an optional element which
1197 value is an integer expression. If present, the WS-HumanTask Definition MUST specify a value
1198 between 0 and 10, where 0 is the highest priority and 10 is the lowest. If not present, the priority
1199 of the task is considered as 5. The result of the expression evaluation is of type
1200 `htt:tPriority`. The `expressionLanguage` attribute specifies the language used in the
1201 expression. The attribute is optional. If not specified, the default language as inherited from the
1202 closest enclosing element that specifies the attribute is used.

- 1203 • `peopleAssignments`: This element is used to specify people assigned to different generic
1204 human roles, i.e. potential owners, and business administrator. The element is optional. See
1205 section 3.5 for more details on people assignments.
 - 1206 • `completionBehavior`: This element is used to specify completion conditions of the task. It is
1207 optional. See section 4.8 for more details on completion behavior.
 - 1208 • `delegation`: This element is used to specify constraints concerning delegation of the task.
1209 Attribute `potentialDelegates` defines to whom the task can be delegated. One of the
1210 following values MUST be specified:
 - 1211 ▪ `anybody`: It is allowed to delegate the task to anybody
 - 1212 ▪ `potentialOwners`: It is allowed to delegate the task to potential owners
1213 previously selected
 - 1214 ▪ `other`: It is allowed to delegate the task to other people, e.g. authorized owners.
1215 The element `<from>` is used to determine the people to whom the task can be
1216 delegated.
 - 1217 ▪ `nobody`: It is not allowed to delegate the task.
- 1218 The delegation element is optional. If this element is not present the task is allowed to be
1219 delegated to anybody.
- 1220 • `presentationElements`: This element is used to specify different information used to display
1221 the task in a task list, such as name, subject and description. See section 4.3 for more details on
1222 presentation elements. The element is optional.
 - 1223 • `outcome`: This optional element identifies the field (of an xsd simple type) in the output message
1224 which reflects the business result of the task. A conversion takes place to yield an outcome of
1225 type `xsd:string`. The optional attribute `queryLanguage` specifies the language used for
1226 selection. If not specified, the default language as inherited from the closest enclosing element
1227 that specifies the attribute is used.
 - 1228 • `searchBy`: This optional element is used to search for task instances based on a custom search
1229 criterion. The result of the expression evaluation is of type `xsd:string`. The
1230 `expressionLanguage` attribute specifies the language used in the expression. The attribute is
1231 optional. If not specified, the default language as inherited from the closest enclosing element that
1232 specifies the attribute is used.
 - 1233 • `rendering`: This element is used to specify the rendering method. It is optional. If not present,
1234 task rendering is implementation dependent. See section 4.4 for more details on rendering tasks.
 - 1235 • `deadlines`: This element specifies different deadlines. It is optional. See section 4.9 for more
1236 details on timeouts and escalations.
 - 1237 • `composition`: This element is used to specify subtasks of a composite task. It is optional. See
1238 section 4.6 for more details on composite tasks.

1239 4.3 Presentation Elements

1240 Information about human tasks or notifications needs to be made available in a human-readable way to
1241 allow users dealing with their tasks and notifications via a user interface, which could be based on various
1242 technologies, such as Web browsers, Java clients, Flex-based clients or .NET clients. For example, a
1243 user queries for her tasks, getting a list of tasks she could work on, displaying a short description of each
1244 task. Upon selection of one of the tasks, more complete information about the task is displayed by the
1245 user interface.

1246 Alternatively, a task or notification could be sent directly to a user's inbox, in which case the same
1247 information would be used to provide a human readable rendering there.

1248 The same human readable information could also be used in reports on all the human tasks executed by
1249 a particular human task management system.

1250 Human readable information can be specified in multiple languages.

1251 **Syntax:**

```
1252 <htd:presentationElements>
1253
1254   <htd:name xml:lang="xsd:language"? >*
1255     Text
1256   </htd:name>
1257
1258   <!-- For the subject and description only,
1259     replacement variables can be used. -->
1260   <htd:presentationParameters expressionLanguage="anyURI"? >?
1261     <htd:presentationParameter name="NCName" type="QName">+
1262       expression
1263     </htd:presentationParameter>
1264   </htd:presentationParameters>
1265
1266   <htd:subject xml:lang="xsd:language"? >*
1267     Text
1268   </htd:subject>
1269
1270   <htd:description xml:lang="xsd:language"?
1271     contentType="mimeTypeString"? >*
1272     <xsd:any minOccurs="0" maxOccurs="unbounded" />
1273   </htd:description>
1274
1275 </htd:presentationElements>
```

1276 **Properties**

1277 The following attributes and elements are defined for the `htd:presentationElements` element.

- 1278 • `name`: This element is the short title of a task. It uses `xml:lang`, a standard XML attribute, to
1279 define the language of the enclosed information. This attribute uses tags according to RFC 1766
1280 (see [RFC1766]). There could be zero or more `name` elements. A WS-HumanTask Definition
1281 MUST NOT specify multiple `name` elements having the same value for attribute `xml:lang`.
- 1282 • `presentationParameters`: This element specifies parameters used in presentation elements
1283 `subject` and `description`. Attribute `expressionLanguage` identifies the expression
1284 language used to define parameters. This attribute is optional. If not specified, the default
1285 language as inherited from the closest enclosing element that specifies the attribute is used.
1286 Element `presentationParameters` is optional and if present then the WS-HumanTask
1287 Definition MUST specify at least one element `presentationParameter`. Element
1288 `presentationParameter` has attribute `name`, which uniquely identifies the parameter
1289 definition within the `presentationParameters` element, and attribute `type` which defines its
1290 type. A WS-HumanTask Definition MUST specify parameters of XSD simple types. When a
1291 `presentationParameter` is used within `subject` and `description`, the syntax is
1292 `{$parameterName}`. The pair "`{{`" represents the character "`{`" and the pair "`}}`" represents
1293 the character "`}`". Only the defined presentation parameters are allowed, that is, a WS-
1294 HumanTask Definition MUST NOT specify arbitrary expressions embedded in this syntax.
- 1295 • `subject`: This element is a longer text that describes the task. It uses `xml:lang` to define the
1296 language of the enclosed information. There could be zero or more `subject` elements. A WS-
1297 HumanTask Definition MUST NOT specify multiple `subject` elements having the same value for
1298 attribute `xml:lang`.
- 1299 • `description`: This element is a long description of the task. It uses `xml:lang` to define the
1300 language of the enclosed information. The optional attribute `contentType` uses content types

1301 according to RFC 2046 (see [RFC 2046]). The default value for this attribute is "text/plain". A WS-
1302 HumanTask Processor MUST support the content type "text/plain". The WS-HumanTask
1303 Processor SHOULD support HTML (such as "text/html" or "application/xml+xhtml"). There could
1304 be zero or more description elements. As descriptions can exist with different content types, it
1305 is allowed to specify multiple description elements having the same value for attribute
1306 xml:lang, but the WS-HumanTask Definition MUST specify different content types.

1307 **Example:**

```
1308 <htd:presentationElements>
1309
1310   <htd:name xml:lang="en-US">Approve Claim</htd:name>
1311   <htd:name xml:lang="de-DE">
1312     Genehmigung der Schadensforderung
1313   </htd:name>
1314
1315   <htd:presentationParameters>
1316     <htd:presentationParameter name="firstname" type="xsd:string">
1317       htd:getInput("ClaimApprovalRequest")/cust/firstname
1318     </htd:presentationParameter>
1319     <htd:presentationParameter name="lastname" type="xsd:string">
1320       htd:getInput("ClaimApprovalRequest")/cust/lastname
1321     </htd:presentationParameter>
1322     <htd:presentationParameter name="euroAmount" type="xsd:double">
1323       htd:getInput("ClaimApprovalRequest")/amount
1324     </htd:presentationParameter>
1325   </htd:presentationParameters>
1326
1327   <htd:subject xml:lang="en-US">
1328     Approve the insurance claim for €{$euroAmount} on behalf of
1329     {$firstname} {$lastname}
1330   </htd:subject>
1331   <htd:subject xml:lang="de-DE">
1332     Genehmigung der Schadensforderung über €{$euroAmount} für
1333     {$firstname} {$lastname}
1334   </htd:subject>
1335
1336   <htd:description xml:lang="en-US" contentType="text/plain">
1337     Approve this claim following corporate guideline #4711.0815/7 ...
1338   </htd:description>
1339   <htd:description xml:lang="en-US" contentType="text/html">
1340     <p>
1341       Approve this claim following corporate guideline
1342       <b>#4711.0815/7</b>
1343       ...
1344     </p>
1345   </htd:description>
1346   <htd:description xml:lang="de-DE" contentType="text/plain">
1347     Genehmigen Sie diese Schadensforderung entsprechend Richtlinie Nr.
1348     4711.0815/7 ...
1349   </htd:description>
1350   <htd:description xml:lang="de-DE" contentType="text/html">
1351     <p>
1352       Genehmigen Sie diese Schadensforderung entsprechend Richtlinie
1353       <b>Nr. 4711.0815/7</b>
1354       ...
1355     </p>
1356   </htd:description>
```

1357
1358 `</htd:presentationElements>`

1359 **4.4 Task Possible Outcomes**

1360 The `<possibleOutcomes>` element provides a way for a task to define which values are usable for the
1361 outcome value of a task. Having a separate definition allows a tool for building tasks to provide support
1362 that understands exactly which outcomes are possible for a particular task.

```
1363 <htd:possibleOutcomes>  
1364   <htd:possibleOutcome name="NCName">+  
1365     <htd:outcomeName xml:lang="xsd:language"?>+  
1366       Language specific display  
1367     </htd:outcomeName>  
1368   </htd:possibleOutcome>  
1369 </htd:possibleOutcomes>
```

1370 Each `<possibleOutcome>` element represents one possible outcome. For the typical example of an
1371 expense report approval, the two outcomes might be 'Approve' and 'Reject'. In addition to the other data
1372 being collected by the rendering in the WS-HumanTask Client, this represents the most important
1373 information about how to proceed in a process that contains multiple tasks. Therefore, a rendering and
1374 client using HTML might choose to show this as a dropdown list, list box with single selection, a set of
1375 submit buttons, or a radio button group.

1376 For each `<possibleOutcome>`, it is possible to have an `<outcomeName>` element to specify a per-
1377 language display name. It uses `xml:lang`, a standard XML attribute, to define the language of the
1378 enclosed information. This attribute uses tags according to RFC 1766 (see [RFC1766]). There could be
1379 zero or more `<outcomeName>` elements. A `<possibleOutcome>` MUST NOT specify multiple
1380 `<outcomeName>` elements having the same value for attribute `xml:lang`.

1381 **4.5 Elements for Rendering Tasks**

1382 Human tasks and notifications need to be rendered on user interfaces like forms clients, portlets, e-mail
1383 clients, etc. The rendering element provides an extensible mechanism for specifying UI renderings for
1384 human tasks and notifications (task-UI). The element is optional. One or more rendering methods can be
1385 provided in a task definition or a notification definition. A task or notification can be deployed on any WS-
1386 HumanTask Processor, irrespective of the fact whether the implementation supports specified rendering
1387 methods or not. The rendering method is identified using a QName.

1388 Unlike for presentation elements, language considerations are opaque for the rendering element because
1389 the rendering applications typically provide multi-language support. Where this is not the case, providers
1390 of certain rendering types can decide to extend the rendering method in order to provide language
1391 information for a given rendering.

1392 The content of the rendering element is not defined by this specification. For example, when used in the
1393 rendering element, XPath extension functions as defined in section 7.2 MAY be evaluated by a WS-
1394 HumanTask Processor.

1395

1396

1397 **Syntax:**

```
1398 <htd:renderings>
1399   <htd:rendering type="QName">+
1400     <xsd:any minOccurs="1" maxOccurs="1" />
1401   </htd:rendering>
1402 </htd:renderings>
```

1403 4.6 Elements for Composite Tasks

1404 A composite task is defined as a `<htd:task>` element with the `<htd:composition>` element enclosed
1405 in it. The following are attributes and elements defined for the `composition` element.

- 1406 • `type`: This optional attribute specifies the order in which enclosed sub-tasks are executed. If the
1407 value is set to “sequential” the sub-tasks MUST be executed in lexical order. Otherwise they
1408 MUST be executed in parallel. The default value for this attribute is “sequential”.
- 1409 • `instantiationPattern`: This optional attribute specifies the way how sub-tasks are
1410 instantiated. If the value is set to “manual” the task client triggers instantiation of enclosed sub-
1411 tasks. Otherwise, they are automatically instantiated at the time the composite task itself turns
1412 into status “inProgress”. The default value for this attribute is “manual”.
- 1413 • `subtask`: This element specifies a task that will be executed as part of the composite task
1414 execution. The `composition` element MUST enclose at least one `subtask` element. The
1415 `subtask` element has the following attributes and elements. The `name` attribute specifies the
1416 name of the sub-task. The name MUST be unique among the names of all sub-tasks within the
1417 `composition` element. The `htd:task` element is used to define the task inline. The
1418 `htd:localTask` element is used to reference a task that will be executed as a sub-task. The
1419 `htd:localTask` element MAY define values for standard overriding attributes: `priority` and
1420 `people assignments`. The `toParts` element is used to assign values to input message of the
1421 sub-task. The enclosed XPath expression MAY refer to the input message of the composite task
1422 or the output message of other sub-task enclosed in the same `composition` element. The
1423 `part` attribute refers to a part of the WSDL message type of the message used in the XPath.
1424 The `expressionLanguage` attribute specifies the expression language used in the enclosing
1425 elements. The default value for this attribute is `urn:ws-ht:sublang:xpath1.0` which
1426 represents the usage of XPath 1.0 within human interactions definition. A WS-HumanTask
1427 Definition that uses expressions MAY override the default expression language for individual
1428 expressions.

1429 When `composition` is defined on a task, the `composition` MUST be applied for each of the potential
1430 owners defined in the task’s `people assignment`.

1431 **Syntax:**

```
1432 <htd:task>
1433   ...
1434   <htd:composition type="sequential|parallel"
1435     instantiationPattern="manual|automatic">
1436     <htd:subtask name="NCName">+
1437
1438     ( <htd:task>
1439       ...
1440     </htd:task>
1441
1442     | <htd:localTask reference="QName">
1443       standard-overriding-elements
1444       ...
1445     </htd:localTask>
```



```

1446 )
1447
1448     <htd:toParts>?
1449         <htd:toPart part="NCName" expressionLanguage="anyURI">+
1450             XPath expression
1451         </htd:toPart>
1452     </htd:toParts>
1453
1454     </htd:subtask>
1455
1456 </htd:composition>
1457 ...
1458 </htd:task>

```

1459 *Standard-overriding-elements* is used in the syntax above as a shortened form of the following list of
1460 elements:

```

1461 <htd:priority expressionLanguage="anyURI"? >
1462     integer-expression
1463 </htd:priority>
1464
1465 <htd:peopleAssignments>?
1466     <htd:genericHumanRole>
1467         <htd:from>...</htd:from>
1468     </htd:genericHumanRole>
1469 </htd:peopleAssignments>

```

1470 4.7 Elements for People Assignment

1471 The <peopleAssignments> element is used to assign people to a task. For each generic human role, a
1472 people assignment element can be specified. A WS-HumanTask Definition MUST specify a people
1473 assignment for potential owners of a human task. An empty <potentialOwners> element is used to
1474 specify that no potential owner is assigned by the human task's definition but another means is used e.g.
1475 nomination. Specifying people assignments for task stakeholders, task initiators, excluded owners and
1476 business administrators is optional. Human tasks never specify recipients. A WS-HumanTask Definition
1477 MUST NOT specify people assignments for actual owners.

1478 Syntax:

```

1479 <htd:peopleAssignments>
1480
1481     <htd:potentialOwners>
1482         ...
1483     </htd:potentialOwners>
1484
1485     <htd:excludedOwners>?
1486         ...
1487     </htd:excludedOwners>
1488
1489     <htd:taskInitiator>?
1490         ...
1491     </htd:taskInitiator>
1492
1493     <htd:taskStakeholders>?
1494         ...
1495     </htd:taskStakeholders>
1496
1497     <htd:businessAdministrators>?
1498         ...
1499     </htd:businessAdministrators>

```

1500
1501 `</htd:peopleAssignments>`

1502 People assignments can result in a set of values or an empty set. In case people assignment results in an
1503 empty set then the task potentially requires administrative attention. This is out of scope of the
1504 specification, except for people assignments for potential owners (see section 4.10.1 “Normal processing
1505 of a Human Task” for more details).

1506 **Example:**

```
1507 <htd:peopleAssignments>
1508   <htd:potentialOwners>
1509     <htd:from logicalPeopleGroup="regionalClerks">
1510       <htd:argument name="region">
1511         htd:getInput("ClaimApprovalRequest")/region
1512       </htd:argument>
1513     </htd:from>
1514   </htd:potentialOwners>
1515
1516   <htd:businessAdministrators>
1517     <htd:from logicalPeopleGroup="regionalManager">
1518       <htd:argument name="region">
1519         htd:getInput("ClaimApprovalRequest")/region
1520       </htd:argument>
1521     </htd:from>
1522   </htd:businessAdministrators>
1523 </htd:peopleAssignments>
```

1524 **4.7.1 Routing Patterns**

1525 Tasks can be assigned to people in sequence and parallel. Elements `htd:sequence` and
1526 `htd:parallel` elements in `htd:potentialOwners` are used to represent such assignments.

1527 **4.7.1.1 Parallel Pattern**

1528 A task can be assigned to people in parallel using the `htd:parallel` element. . The `htd:parallel`
1529 element is defined as follows:

- 1530 • The `htd:from` element defines the parallel potential owners. This can evaluate to multiple
1531 users/groups.
- 1532 • The attribute ‘type’ in `htd:parallel` identifies how parallel assignments are created for the
1533 multiple users/groups returned from `htd:from`. If type is ‘all’ then an assignment MUST be
1534 created for each user returned by `htd:from`. If type is ‘single’ then an assignment MUST be
1535 created for each `htd:from` clause (this assignment could have with n potential owners). The
1536 default value of type is ‘all’.
- 1537 • The `htd:parallel` and `htd:sequence` elements define nested routing patterns within the
1538 parallel routing pattern
- 1539 • The `htd:completionBehavior` defines when the routing pattern completes. The completion
1540 criteria also define how the result is constructed for the parent task when a parallel routing
1541 pattern is complete.

1542 Each parallel assignment MUST result in a separate sub task. Sub tasks created for each parallel
1543 assignment MUST identify the parent task using the `htd:parentTaskId`.

1544

1545

1546 **Syntax:**

```
1547 <htd:potentialOwners>
1548   <htd:parallel type="all|single"?>
1549     <htd:completionBehavior>
1550     <htd:from>...</htd:from>*
1551     pattern*
1552   </htd:parallel>
1553 </htd:potentialOwners>
```

1554 **Example:**

```
1555 <htd:peopleAssignments>
1556   <htd:potentialOwners>
1557     <htd:parallel type="all">
1558       <htd:from>
1559         htd:getInput("ClaimApprovalRequest")/claimAgent
1560       </htd:from>
1561     </htd:parallel>
1562   </htd:potentialOwners>
1563 </htd:peopleAssignments>
```

1564 **4.7.1.2 Sequential Pattern**

1565 A task can be assigned to people in sequence using the `htd:sequence` element. The `htd:sequence`
1566 is defined as follows:

- 1567
- 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.

1578 Sequential routing patterns MUST use a separate sub task for each step in a sequential pattern. Sub
1579 tasks created for each sequential assignment MUST identify the parent task using the
1580 `htd:parentTaskId`.

1581 **Syntax:**

```
1582 <htd:potentialOwners>
1583   <htd:sequence type="all|single"?>
1584     <htd:completionBehavior?>
1585     <htd:from>...</htd:from>*
1586     pattern*
1587   </htd:sequence>
1588 </htd:potentialOwners>
```

1589

1590 **Example:**

```
1591 <htd:peopleAssignments>
1592   <htd:potentialOwners>
1593     <htd:sequence type="all">
1594       <htd:from logicalPeopleGroup="regionalClerks">
1595         <htd:argument name="region">
1596           htd:getInput("ClaimApprovalRequest")/region
1597         </htd:argument>
1598       </htd:from>
1599       <htd:from logicalPeopleGroup="regionalManager">
1600         <htd:argument name="region">
1601           htd:getInput("ClaimApprovalRequest")/region
1602         </htd:argument>
1603       </htd:from>
1604     </htd:sequence>
1605   </htd:potentialOwners>
1606 </htd:peopleAssignments/>
```

1607 **4.8 Completion Behavior**

1608 The completion behavior of a task, routing pattern or composite task can be influenced by a specification
1609 of completion conditions and the result construction for tasks, routing patterns, or composite tasks. For
1610 this purpose, the task, routing pattern or composite task contains a `htd:completionBehavior`
1611 element.

1612 Multiple completion conditions can be specified as nested `htd:completion` elements. They are
1613 evaluated in lexical order. When one of the specified completion conditions is met then the task is
1614 considered to be completed; in case of routing patterns and composite tasks all remaining running sub
1615 tasks **MUST** be skipped (i.e., set to the "Obsolete" state) and the associated result construction **MUST** be
1616 applied.

1617 In case of composite tasks and routing patterns the following applies: At most one default completion
1618 **MUST** be specified with no completion condition in order to specify the result construction after regular
1619 completion of all sub tasks. If no result construction is applied, e.g. because no "default" result
1620 construction is specified and none of the specified completion conditions is met, then the parent task's
1621 output is not created, i.e., it remains uninitialized. Moreover, note that a completion condition can be
1622 specified without referencing sub task output data, which allows the parent task to be considered
1623 completed even without creating any sub tasks. When output data from sub tasks is referenced by
1624 completion conditions or result constructions, only output data of already finished sub tasks **MUST** be
1625 considered.

1626 If none of the specified completion conditions is met then the state of the task or the parent task remains
1627 unchanged.

```
1628 <htd:completionBehavior completionAction="manual|automatic"?>?
1629   <htd:completion name="NCName">*
1630     <htd:condition ... >
1631       ...
1632     </htd:condition>
1633     <htd:result>?
1634     ...
1635     <htd:result>
1636   </htd:completion>
1637   <htd:defaultCompletion>?
1638     <htd:result>
1639     ...
1640     <htd:result>
1641   </htd:defaultCompletion>
1642 </htd:completionBehavior>
```

1643 The `completionBehavior` element has optional attribute `completionAction`. This optional
1644 attribute specifies how the task, routing pattern, or composite task is completed. If the value is set to
1645 "manual" the task or parent task MUST be completed explicitly by the actual owner as soon as the
1646 completion conditions evaluate to true. If the value is set to "automatic" the task or parent task MUST be
1647 set to complete as soon as the completion conditions evaluate to true. For routing patterns, the
1648 `completionAction` attribute MUST have value "automatic". The default value for this attribute is
1649 "automatic".

1650 If `completionBehavior` is not specified, the default behavior is that of a `completionBehavior` with
1651 `completionCondition` is "true" and a `completionAction` of "manual" for simple and composite
1652 tasks, and "automatic" for routing patterns.

1653 4.8.1 Completion Conditions

1654 A completion condition defines when a task or a set of sub tasks associated with the parent task is
1655 considered completed. It is specified Boolean expression which can refer to input data of the task, the
1656 parent task or its sub tasks, output data produced by already finished sub tasks, or other data obtained
1657 from WS-HumanTask API calls (e.g. the number of sub tasks), or functions that test that some designated
1658 amount of time has passed.

1659 The completion condition MUST be defined using an `htd:condition` element.

```
1660 <htd:condition expressionLanguage="anyURI"?>  
1661   boolean expression  
1662 </htd:condition>
```

1663 Within the Boolean expression of a completion condition, aggregation functions can be used to evaluate
1664 output data produced by the already finished sub tasks of the parent task.

1665 If an error (e.g. division by zero) occurs during the condition evaluation then the condition MUST be
1666 considered to have evaluated to "false".

1667 The time functions that are available are defined as follows:

- 1668 • `boolean htd:waitFor(string)`
 - 1669 ○ The parameter is an XPath expression evaluating to a string conforming to the definition
1670 ○ of the XML Schema type `duration`
 - 1671 ○ The return value is `true` after the specified duration has elapsed, otherwise `false`
- 1672 • `boolean htd:waitUntil(string)`
 - 1673 ○ The parameter is an XPath expression evaluating to a string conforming to the definition
1674 ○ of the XML Schema type `dateTime`

1675 The return value is `true` after the specified absolute time has passed, otherwise `false`.

1676 Completion conditions of a task MUST use only time functions.

1677 4.8.1.1 Evaluating the Completion Condition

1678 The time functions in the completion condition are be evaluated with respect to the beginning of execution
1679 of the task or parent task on which the completion is defined. To achieve this, the evaluation of the
1680 `htd:waitFor` and `htd:waitUntil` calls within the condition are treated differently from the rest of the
1681 expression. When the containing task or parent task is created, the actual parameter expression for any
1682 `htd:waitFor` and `htd:waitUntil` calls MUST be evaluated and the completion condition should be
1683 rewritten to replace these calls with only `htd:waitUntil` calls with constant parameters. The durations
1684 calculated for any `htd:waitFor` calls MUST be converted into absolute times and rewritten as
1685 `htd:waitUntil` calls. The result of these replacements is called the *preprocessed completion*
1686 *condition*.

1687

1688

1689 For the parent task, the preprocessed completion condition MUST be evaluated at the following times:

- 1690 • Before starting the first subtask (it may be complete before it starts)
- 1691 • Whenever a subtask completes
- 1692 • Whenever a duration specified in a `htd:waitFor` call has elapsed
- 1693 • Whenever an absolute time specified in a `htd:waitUntil` call is passed.

1694 For tasks, the preprocessed completion condition MUST be evaluated at the following times:

- 1695 • Before starting the task (it may be complete before it starts)
- 1696 • Whenever a duration specified in a `htd:waitFor` call has elapsed
- 1697 • Whenever an absolute time specified in a `htd:waitUntil` call is passed.

1698 **Example:**

1699 The first completion condition may be met even without starting sub tasks. When both parts of the second
1700 completion condition are met, that is, 7 days have expired and more than half of the finished sub tasks
1701 have an outcome of "Rejected", then the parallel routing pattern is considered completed.

```
1702 <htd:parallel>  
1703   ...  
1704   <htd:completionBehavior>  
1705     <htd:completion>  
1706       <htd:condition>  
1707         htd:getInput("ClaimApprovalRequest")/amount < 1000  
1708       </htd:condition>  
1709       <htd:result> ... </htd:result>  
1710     </htd:completion>  
1711     <htd:completion>  
1712       <htd:condition>  
1713         ( htd:getCountOfSubtasksWithOutcome("Rejected") /  
1714           htd:getCountOfSubtasks() > 0.5 )  
1715         and htd:waitFor("P7D")  
1716       </htd:condition>  
1717       <htd:result> ... </htd:result>  
1718     </htd:completion>  
1719   </htd:completionBehavior>  
1720   ...  
1721 </htd:parallel>
```

1722 **4.8.2 Result Construction from Parallel Subtasks**

1723 When multiple sub tasks are created in order let several people work on their own sub task in parallel
1724 then the outputs of these sub tasks sometimes need to be combined for the creation of the parent task's
1725 output.

1726 If all sub tasks have the same interface definition (as in routing patterns) then the result construction can
1727 be defined in a declarative way using aggregation functions. Alternatively, the result may be created using
1728 explicit assignments.

1729 The result construction MUST be defined as `htd:result` element, containing one or more
1730 `htd:aggregate` or `htd:copy` elements, executed in the order in which they appear in the task
1731 definition.

```
1732 <htd:result>  
1733   (  
1734     <htd:aggregate ... />  
1735     |  
1736     <htd:copy> ... </htd:copy>
```

1737)+
1738 </htd:result>

1739 4.8.2.1 Declarative Result Aggregation

1740 An `htd:aggregate` element describes the result aggregation for a leaf element of the parent task's
1741 output document. In most cases, this approach is only meaningful for routing patterns with identical sub
1742 task interfaces. Note that the construction of (complex-typed) non-leaf elements is out of scope of the
1743 declarative result aggregation.

```
1744 <htd:aggregate part="NCName"?  
1745     location="query"?  
1746     condition="bool-expr"?  
1747     function="function-call"/>+
```

1748 The `htd:aggregate` element is defined as follows:

- 1749 • The optional `part` attribute MUST contain the name of a WSDL part. The `part` attribute MUST be
1750 specified when the task interface is defined using a WSDL message with more than one WSDL
1751 part.
- 1752 • The optional `location` attribute MUST contain a query pointing to the location of a leaf element
1753 of the tasks' output documents:
 - 1754 ○ For each parallel sub task, this is the location of exactly one element of the sub task's
1755 output document that is processed by the aggregation function. Each sub tasks' output
1756 element is (conditionally) added to a node-set passed as parameter to the aggregation
1757 function.
 - 1758 ○ For the parent task, this is the element created in the task's output document that is the
1759 computed return value of the aggregation function.
- 1760 • The optional `condition` attribute MUST contain a Boolean expression evaluated on each sub
1761 task's output document. If the expression evaluates to `true` then the sub task's output element
1762 identified by `location` is added to the node-set passed to the aggregation function.
- 1763 • The mandatory `function` attribute contains the name of the aggregation function (QName; see
1764 a list of supported aggregation functions below) and optional arguments, in the following form:
1765 `FunctionName '(' (Argument (',' Argument)*)? ')'`
1766 Important:
 - 1767 ○ The first parameter of each aggregation function is the node-set of sub task's output
1768 elements to be aggregated. This parameter is inserted implicitly and MUST NOT be
1769 specified within the `function` attribute.
 - 1770 ○ Within the `function` attribute, function arguments MUST be specified only for *additional*
1771 parameters defined for an aggregation function.

1772 If a declarative result aggregation is applied, it is still possible that no values can be provided for the
1773 aggregation of a particular output field, for example, if no subtask has set a value to an optional field (by
1774 omission or by an explicit nil value).

1775 In this case, the following rules determine how the aggregated output field of the parent task is set.

- 1776 • Rule (1): If the result value is optional (element defined with `minOccurs="0"` or attribute defined
1777 with `use="optional"`) then the corresponding element or attribute in the parent task output
1778 MUST be omitted.
- 1779 • Rule (2): If rule (1) does not apply and a default value is provided (element or attribute defined
1780 with `default="{value}"`) then the parent task output element or attribute MUST be explicitly
1781 set to this default value.
- 1782 • Rule (3): If rules (1)-(2) do not apply and the result value is a nillable element (element defined
1783 with `nillable="true"`) then the parent task output element MUST be set to a nil value (<a
1784 `xsi:nil="true"/>`).

- Rule (4): If rules (1)-(3) do not apply, that is, the result is mandatory (element defined with minOccurs="1" or attribute defined with use="required") but a value cannot be supplied, then a standard fault htd:aggregationFailure MUST be thrown to indicate a non-recoverable error.

1789 **Example:**

1790 Consider the following output document used in a parallel routing pattern:

```

1791 <element name="Award" type="tns:tAward" />
1792 <complexType name="tAward">
1793   <sequence>
1794     <element name="AwardRecommended" type="xsd:string" />
1795     <element name="AwardDetails" type="tns:tAwardDetails" />
1796   </sequence>
1797 </complexType>
1798 <complexType name="tAwardDetails">
1799   <sequence>
1800     <element name="Amount" type="xsd:integer" />
1801     <element name="Appraisal" type="xsd:string" />
1802   </sequence>
1803 </complexType>

```

1804 A possible result aggregation could then look like this. The first aggregation determines the most frequent occurrence of an award recommendation. The second aggregation calculates the average award amount for sub tasks with an award recommendation of 'yes'. The third aggregation creates a comma-separated concatenation of all sub task's appraisals.

```

1808 <htd:parallel ...>
1809   ...
1810   <htd:completionBehavior>
1811     <htd:completion>
1812       <htd:condition> ... </htd:condition>
1813       <htd:result>
1814         <htd:aggregate location="/Award/AwardRecommended"
1815           function="htd:mostFrequentOccurence()" />
1816         <htd:aggregate location="/Award/AwardDetails/Amount"
1817           condition="/Award/AwardRecommended='yes'"
1818           function="htd:avg()" />
1819         <htd:aggregate location="/Award/AwardDetails/Appraisal"
1820           function="htd:concatWithDelimiter(',')" />
1821       </htd:result>
1822     </htd:completion>
1823   </htd:completionBehavior>
1824 </htd:parallel>

```

1825 **4.8.2.2 Explicit Result Assignment**

1826 An htd:copy element describes the explicit assignment to an element of the parent task's output document.

```

1828 <htd:copy>+
1829   <htd:from expressionLanguage="anyURI"?>
1830     expression
1831   </htd:from>
1832   <htd:to queryLanguage="anyURI"?>
1833     query
1834   </htd:to>
1835 </htd:copy>

```

1836 The htd:copy element is defined as follows:

- 1837 • The mandatory `htd:from` element MUST contain an expression used to calculate the result
1838 value. The expression can make use of WS-HumanTask aggregation functions.
- 1839 • The mandatory `htd:to` element MUST contain a query pointing to the location of an element of
1840 the tasks' output documents. This is the element created in the task's output document.

1841 **Example 1:**

1842 Consider the following output document used in a parallel routing pattern:

```
1843 <element name="Order" type="tns:tOrder" />
1844 <complexType name="tOrder">
1845   <sequence>
1846     <element name="Item" type="tns:tItem" maxOccurs="unbounded"/>
1847     <element name="TotalPrice" type="xsd:integer" />
1848   </sequence>
1849 </complexType>
1850 <complexType name="tItem">
1851   <sequence>
1852     ...
1853   </sequence>
1854 </complexType>
```

1855 A possible result aggregation could then look like this. All sub task order item lists are concatenated to
1856 one parent task order item list. The total price is calculated using an aggregation function.

```
1857 <htd:parallel>
1858   ...
1859   <htd:completionBehavior>
1860     <htd:completion>
1861       <htd:condition> ... </htd:condition>
1862       <htd:result>
1863         <htd:copy>
1864           <htd:from>
1865             htd:getSubtaskOutputs("orderResponse", "/Order/Item")
1866           </htd:from>
1867           <htd:to>/Order/Item</htd:to>
1868         </htd:copy>
1869         <htd:copy>
1870           <htd:from>
1871             htd:sum(htd:getSubtaskOutputs("orderResponse",
1872                                           "/Order/TotalPrice"))
1873           </htd:from>
1874           <htd:to>/Order/TotalPrice</htd:to>
1875         </htd:copy>
1876       </htd:result>
1877     </htd:completion>
1878   </htd:completionBehavior>
1879 </htd:parallel>
```

1880 **Example 2:**

1881 Output data from heterogeneous sub tasks is assigned into the parent task's output. The complete
1882 complex-typed sub task output documents are copied into child elements of the parent task output
1883 document.

```
1884 <htd:task name="bookTrip">
1885   ... produces itinerary ...
1886
1887   <htd:composition type="parallel" ...>
1888     <htd:subtask name="bookHotel">
1889       <htd:task>
```

```

1890     ... produces hotelReservation ...
1891     </htd:task>
1892 </htd:subtask>
1893 <htd:subtask name="bookFlight">
1894     <htd:task>
1895         ... produces flightReservation ...
1896     </htd:task>
1897 </htd:subtask>
1898 </htd:composition>
1899 ...
1900 <htd:completionBehavior>
1901     <htd:defaultCompletion>
1902         <htd:result>
1903             <htd:copy>
1904                 <htd:from>
1905                     htd:getSubtaskOutput("bookHotel",
1906                                             "bookHotelResponse",
1907                                             "/hotelReservation")
1908                 </htd:from>
1909                 <htd:to>/itinerary/hotelReservation</htd:to>
1910             </htd:copy>
1911             <htd:copy>
1912                 <htd:from>
1913                     htd:getSubtaskOutput("bookFlight",
1914                                             "bookFlightResponse",
1915                                             "/flightReservation")
1916                 </htd:from>
1917                 <htd:to>/itinerary/flightReservation</htd:to>
1918             </htd:copy>
1919         </htd:result>
1920     </htd:defaultCompletion>
1921 </htd:completionBehavior>
1922 </htd:task>

```

1923 4.9 Elements for Handling Timeouts and Escalations

1924 Timeouts and escalations allow the specification of a date or time before which the task or sub task has to
1925 reach a specific state. If the timeout occurs a set of actions is performed as the response. The state of the
1926 task is not changed. Several deadlines are specified which differ in the point when the timer clock starts
1927 and the state which has to be reached with the given duration or by the given date. They are:

- 1928 • Start deadline: Specifies the time until the task has to start, i.e. it has to reach state *InProgress*. It
1929 is defined as either the period of time or the point in time until the task has to reach state
1930 *InProgress*. Since expressions are allowed, durations and deadlines can be calculated at runtime,
1931 which for example enables custom calendar integration. The time starts to be measured from the
1932 time at which the task enters the state *Created*. If the task does not reach state *InProgress* by the
1933 deadline an escalation action or a set of escalation actions is performed. Once the task is started,
1934 the timer becomes obsolete.
- 1935 • Completion deadline: Specifies the due time of the task. It is defined as either the period of time
1936 until the task gets due or the point in time when the task gets due. The time starts to be measured
1937 from the time at which the task enters the state *Created*. If the task does not reach one of the final
1938 states (*Completed*, *Failed*, *Error*, *Exited*, *Obsolete*) by the deadline an escalation action or a set
1939 of escalation actions is performed.

1940 The element `<deadlines>` is used to include the definition of all deadlines within the task definition. It is
1941 optional. If present then the WS-HumanTask Definition MUST specify at least one deadline. Deadlines

1942 defined in ad-hoc sub tasks created at runtime MUST NOT contradict the deadlines of their parent task.
1943 The value of the name attribute MUST be unique for all deadline specifications within a task definition.

1944 **Syntax:**

```
1945 <htd:deadlines>  
1946  
1947   <htd:startDeadline name="NCName">*  
1948  
1949     <htd:documentation xml:lang="xsd:language"? >*  
1950       text  
1951     </htd:documentation>  
1952  
1953     ( <htd:for expressionLanguage="anyURI"? >  
1954       duration-expression  
1955     </htd:for>  
1956     | <htd:until expressionLanguage="anyURI"? >  
1957       deadline-expression  
1958     </htd:until>  
1959   )  
1960  
1961   <htd:escalation name="NCName">*  
1962     ...  
1963   </htd:escalation>  
1964  
1965 </htd:startDeadline>  
1966  
1967 <htd:completionDeadline name="NCName">*  
1968   ...  
1969 </htd:completionDeadline>  
1970  
1971 </htd:deadlines>
```

1972 The language used in expressions is specified using the `expressionLanguage` attribute. This attribute
1973 is optional. If not specified, the default language as inherited from the closest enclosing element that
1974 specifies the attribute is used.

1975 For all deadlines if a status is not reached within a certain time then an escalation action, specified using
1976 element `<escalation>`, can be triggered. The `<escalation>` element is defined in the section below.
1977 When the task reaches a final state (*Completed*, *Failed*, *Error*, *Exited*, *Obsolete*) all deadlines are deleted.

1978 Escalations are triggered if

- 1979 1. The associated point in time is reached, or duration has elapsed, and
- 1980 2. The associated condition (if any) evaluates to true

1981 Escalations use notifications to inform people about the status of the task. Optionally, a task might be
1982 reassigned to some other person or group as part of the escalation. Notifications are explained in more
1983 detail in section 6 "Notifications". For an escalation, a WS-HumanTask Definition MUST specify exactly
1984 one escalation action.

1985 When defining escalations, a notification can be either referred to, or defined inline.

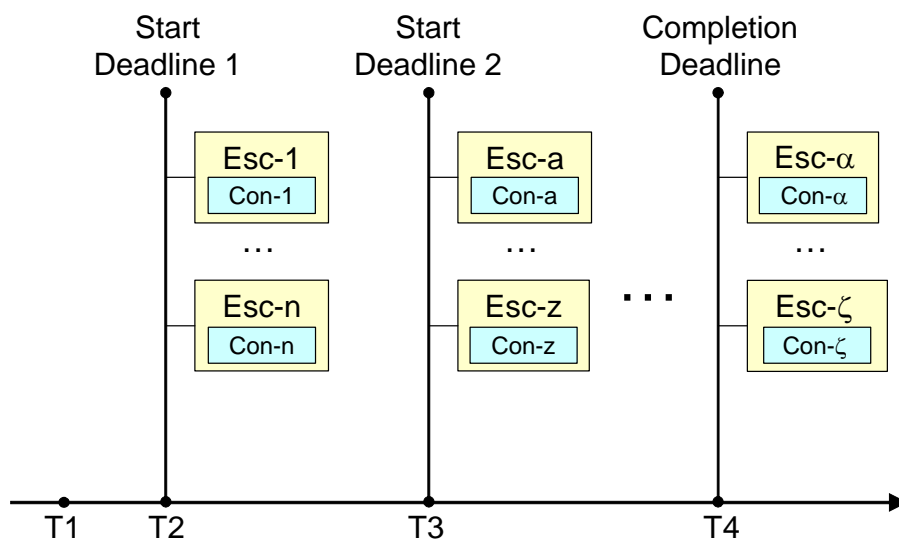
- 1986 • A notification defined in the `<humanInteractions>` root element or imported from a different
1987 namespace can be referenced by specifying its QName in the `reference` attribute of a
1988 `<localNotification>` element. When referring to a notification, the priority and the people
1989 assignments of the original notification definition MAY be overridden using the elements
1990 `<priority>` and `<peopleAssignments>` contained in the `<localNotification>` element.
- 1991 • An inlined notification is defined by a `<notification>` element.

1992 Notifications used in escalations can use the same type of input data as the surrounding task or sub task,
 1993 or different type of data. If the same type of data is used then the input message of the task or sub task is
 1994 passed to the notification implicitly. If not, then the <toPart> elements are used to assign appropriate
 1995 data to the notification, i.e. to explicitly create a multi-part WSDL message from the data. The part
 1996 attribute refers to a part of the WSDL message. The expressionLanguage attribute specifies the
 1997 language used in the expression. The attribute is optional. If not specified, the default language as
 1998 inherited from the closest enclosing element that specifies the attribute is used.

1999 A WS-HumanTask Definition MUST specify a <toPart> element for every part in the WSDL message
 2000 definition because parts not explicitly represented by <toPart> elements would result in uninitialized parts
 2001 in the target WSDL message. The order in which parts are specified is not relevant. If multiple <toPart>
 2002 elements are present, a WS-HumanTask Processor MUST execute them in an “all or nothing” manner. If
 2003 any of the <toPart>s fails, the escalation action will not be performed and the execution of the task is not
 2004 affected.

2005 Reassignments are used to replace the potential owners of a task when an escalation is triggered. The
 2006 <reassignment> element is used to specify reassignment. If present then a WS-HumanTask Definition
 2007 MUST specify potential owners. A reassignment triggered by a sub task escalation MUST apply to the
 2008 sub task only. A reassignment MAY comprise of a complex people assignment using Routing Patterns.

2009 In the case where several reassignment escalations are triggered, the first reassignment (lexical order)
 2010 MUST be considered for execution by the WS-HumanTask Processor. The task is set to state Ready after
 2011 reassignment. Reassignments and notifications are performed in the lexical order.



2012
 2013 A task MAY have multiple start deadlines and completion deadlines associated with it. Each such
 2014 deadline encompasses escalation actions each of which MAY send notifications to certain people. The
 2015 corresponding set of people MAY overlap.

2016 As an example, the figure depicts a task that has been created at time T1. Its two start deadlines would
 2017 be missed at time T2 and T3, respectively. The associated escalations whose conditions evaluate to
 2018 “true” are triggered. Both, the escalations Esc-1 to Esc-n as well as escalations Esc-a to Esc-z can
 2019 involve an overlapping set of people. The completion deadline would be missed at time T4.

2020 **Syntax:**

```
2021 <htd:deadlines>
2022
2023   <htd:startDeadline name="NCName">*
2024   ...
2025   <htd:escalation name="NCName">*
```

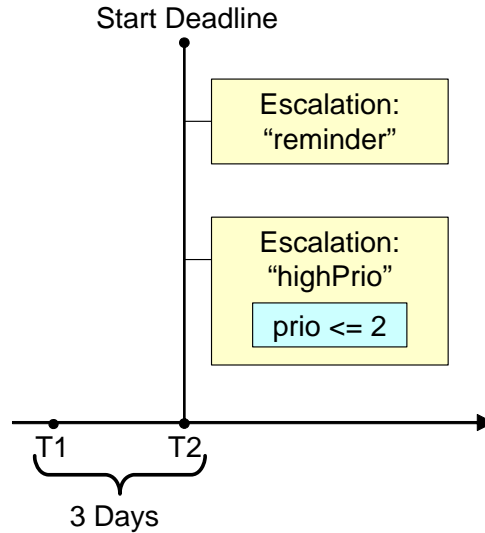
```

2026
2027     <htd:condition expressionLanguage="anyURI"?>?
2028         boolean-expression
2029     </htd:condition>
2030
2031     <htd:toParts>?
2032         <htd:toPart part="NCName"
2033             expressionLanguage="anyURI"?>+
2034             expression
2035         </htd:toPart>
2036     </htd:toParts>
2037
2038     <!-- notification specified by reference -->
2039     <htd:localNotification reference="QName"?
2040         <htd:priority expressionLanguage="anyURI"?>?
2041         integer-expression
2042     </htd:priority>
2043     <htd:peopleAssignments>?
2044         <htd:recipients>
2045             ...
2046         </htd:recipients>
2047     </htd:peopleAssignments>
2048
2049 </htd:localNotification>
2050
2051 <!-- notification specified inline -->
2052 <htd:notification name="NCName"?
2053     ...
2054 </htd:notification>
2055
2056 <htd:reassignment>?
2057
2058     <htd:potentialOwners>
2059         ...
2060     </htd:potentialOwners>
2061
2062 </htd:reassignment>
2063
2064 </htd:escalation>
2065
2066 </htd:startDeadline>
2067
2068 <htd:completionDeadline name="NCName">*
2069     ...
2070 </htd:completionDeadline>
2071
2072 </htd:deadlines>
2073

```

2074 **Example:**

2075 The following example shows the specification of a start deadline with escalations. At runtime, the
2076 following picture depicts the result of what is specified in the example:



2077 The human task is created at T1. If it has not been started, i.e., no person is working on it until T2, then
2078 the escalation “reminder” is triggered that notifies the potential owners of the task that work is waiting for
2079 them. In case the task has high priority then at the same time the regional manager is informed. If the
2080 task amount is greater than or equal 10000 the task is reassigned to Alan.

2081 In case that task has been started before T2 was reached, then the start deadline is deactivated, no
2082 escalation occurs.

```
2083 <htd:startDeadline name="sendNotifications">
2084   <htd:documentation xml:lang="en-US">
2085     If not started within 3 days, - escalation notifications are sent
2086     if the claimed amount is less than 10000 - to the task's potential
2087     owners to remind them or their todo - to the regional manager, if
2088     this approval is of high priority (0,1, or 2) - the task is
2089     reassigned to Alan if the claimed amount is greater than or equal
2090     10000
2091   </htd:documentation>
2092   <htd:for>P3D</htd:for>
2093   <htd:escalation name="reminder">
2094
2095     <htd:condition>
2096       <![CDATA[
2097         htd:getInput("ClaimApprovalRequest")/amount < 10000
2098       ]]>
2099     </htd:condition>
2100
2101     <htd:toParts>
2102       <htd:toPart name="firstname">
2103         htd:getInput("ClaimApprovalRequest","ApproveClaim")/firstname
2104       </htd:toPart>
2105       <htd:toPart name="lastname">
2106         htd:getInput("ClaimApprovalRequest","ApproveClaim")/lastname
2107       </htd:toPart>
2108     </htd:toParts>
2109   </htd:escalation>
2110 </htd:startDeadline>
```



```

2110 <htd:localNotification reference="tns:ClaimApprovalReminder">
2111
2112   <htd:documentation xml:lang="en-US">
2113     Reuse the predefined notification "ClaimApprovalReminder".
2114     Overwrite the recipients with the task's potential owners.
2115   </htd:documentation>
2116
2117   <htd:peopleAssignments>
2118     <htd:recipients>
2119       <htd:from>htd:getPotentialOwners("ApproveClaim")</htd:from>
2120     </htd:recipients>
2121   </htd:peopleAssignments>
2122
2123 </htd:localNotification>
2124
2125 </htd:escalation>
2126
2127 <htd:escalation name="highPrio">
2128
2129   <htd:condition>
2130     <![CDATA[
2131       (htd:getInput("ClaimApprovalRequest")/amount < 10000
2132       && htd:getInput("ClaimApprovalRequest")/prio <= 2)
2133     ]]>
2134   </htd:condition>
2135
2136   <!-- task input implicitly passed to the notification -->
2137
2138   <htd:notification name="ClaimApprovalOverdue">
2139     <htd:documentation xml:lang="en-US">
2140       An inline defined notification using the approval data as its
2141       input.
2142     </htd:documentation>
2143
2144     <htd:interface portType="tns:ClaimsHandlingPT"
2145       operation="escalate" />
2146
2147     <htd:peopleAssignments>
2148       <htd:recipients>
2149         <htd:from logicalPeopleGroup="regionalManager">
2150           <htd:argument name="region">
2151             htd:getInput("ClaimApprovalRequest")/region
2152           </htd:argument>
2153         </htd:from>
2154       </htd:recipients>
2155     </htd:peopleAssignments>
2156
2157     <htd:presentationElements>
2158       <htd:name xml:lang="en-US">Claim approval overdue</htd:name>
2159       <htd:name xml:lang="de-DE">
2160         Überfällige Schadensforderungsgenehmigung
2161       </htd:name>
2162     </htd:presentationElements>
2163
2164   </htd:notification>
2165
2166 </htd:escalation>
2167

```

```

2168 <htd:escalation name="highAmountReassign">
2169
2170   <htd:condition>
2171     <![CDATA[
2172       htd:getInput("ClaimApprovalRequest")/amount >= 10000
2173     ]]>
2174   </htd:condition>
2175
2176   <htd:reassignment>
2177     <htd:documentation>
2178       Reassign task to Alan if amount is greater than or equal
2179       10000.
2180     </htd:documentation>
2181
2182     <htd:potentialOwners>
2183       <htd:from>
2184         <htd:literal>
2185           <htt:organizationalEntity>
2186             <htt:user>Alan</htt:user>
2187           </htt:organizationalEntity>
2188         </htd:literal>
2189       </htd:from>
2190     </htd:potentialOwners>
2191
2192   </htd:reassignment>
2193
2194 </htd:escalation>
2195
2196 </htd:startDeadline>

```

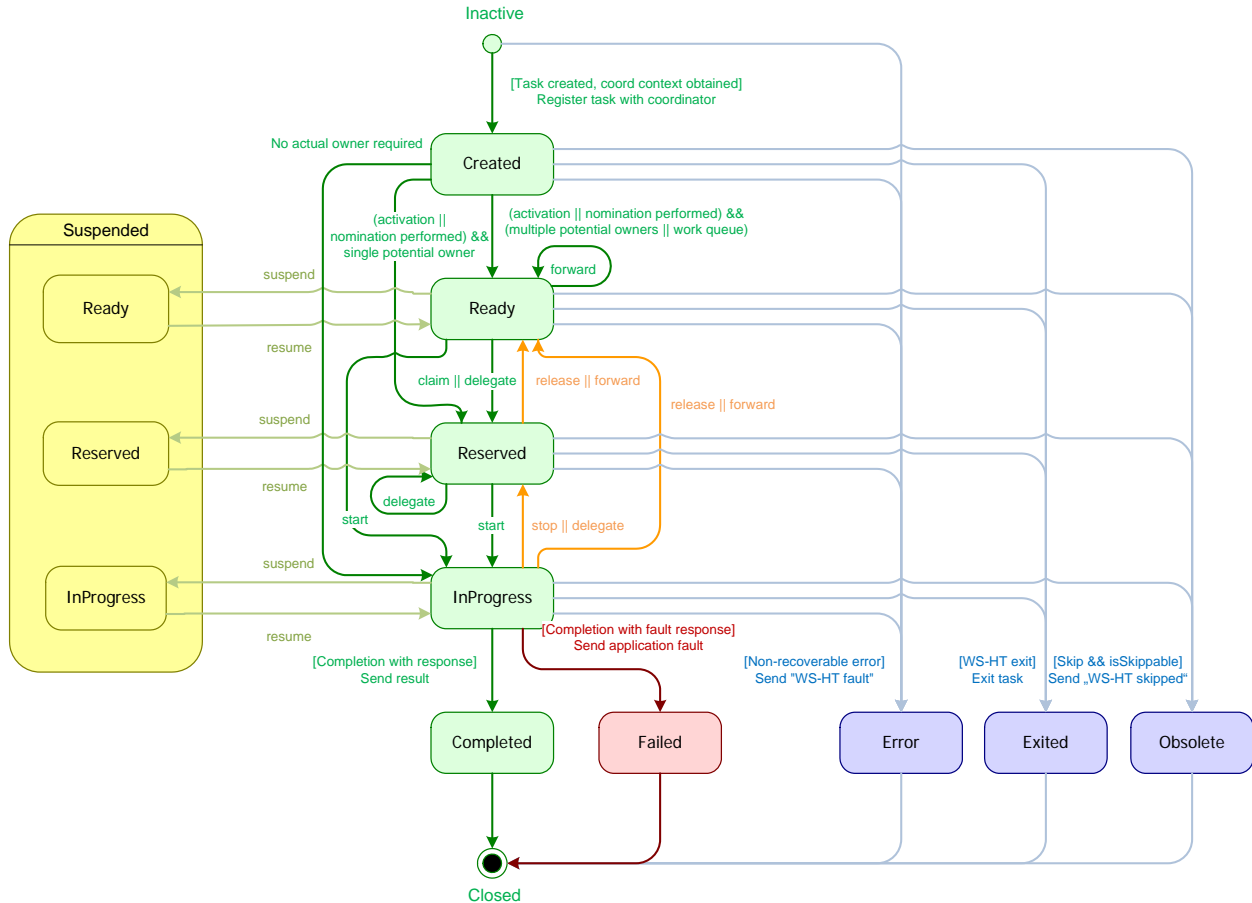
2197 All timeouts and escalations apply to sub tasks also. If htd:escalation is triggered for a sub task, then any
2198 htd:reassignment MUST be applied only to that.

2199

2200

2201 4.10 Human Task Behavior and State Transitions

2202 Human tasks can have a number of different states and substates. The state diagram for human tasks
2203 below shows the different states and the transitions between them.



2204

2205 4.10.1 Normal processing of a Human Task

2206 Upon creation, a task goes into its initial state *Created*. Task creation starts with the initialization of its
2207 properties in the following order:

- 2208 1. Input message
- 2209 2. Priority
- 2210 3. Generic human roles (such as excluded owners, potential owners and business administrators)
- 2211 are made available in the lexical order of their definition in the people assignment definition with
- 2212 the constraint that excluded owners are taken into account when evaluating the potential owners.
- 2213 4. All other properties are evaluated after these properties in an implementation dependent order.

2214 Task creation succeeds irrespective of whether the people assignment returns a set of values or an
2215 empty set. People queries that cannot be executed successfully are treated as if they were returning an
2216 empty set.

2217 If potential owners were not assigned automatically during task creation then they **MUST** be assigned
2218 explicitly using nomination, which is performed by the task's business administrator. The result of
2219 evaluating potential owners removes the excluded owners from results. The task remains in the state
2220 *Created* until it is activated (i.e., an activation timer has been specified) and has potential owners.

2221 When the task has a single potential owner, it transitions into the *Reserved* state, indicating that it is
2222 assigned to a single actual owner. Otherwise (i.e., when it has multiple potential owners or is assigned to
2223 a work queue), it transitions into the *Ready* state, indicating that it can be claimed by one of its potential
2224 owners. Once a potential owner claims the task, it transitions into the *Reserved* state, making that
2225 potential owner the actual owner.

2226 Once work is started on a task that is in state *Ready* or *Reserved*, it goes into the *InProgress* state,
2227 indicating that it is being worked on – if the transition is from *Ready*, the user starting the work becomes
2228 its actual owner.

2229 On successful completion of the work, the task transitions into the *Completed* final state. On unsuccessful
2230 completion of the work (i.e., with an exception), the task transitions into the *Failed* final state.

2231 The lifecycle of sub tasks is the same as that of the main task.

2232 For human tasks that have subtasks two different cases exist, with different implications:

2233 1. Tasks with subtasks where an actual owner is required

2234 2. Tasks with subtasks where no actual owner is required

2235 The first case has the sub-case where a potential owner has been modeled on the primary task and
2236 subtasks have been modeled that are activated either manually or automatically. Another sub-case of the
2237 first case is the one where no potential owner has been modeled and thus nomination has to occur. In all
2238 cases there is an actual owner eventually and the primary task goes through the state transitions from
2239 *Created* to *Ready* to *Reserved* to *InProgress*, etc.

2240 In the second case where no actual owner is desired the human task (the primary task) directly transitions
2241 from state *Created* to *InProgress*. Subtasks are always instantiated automatically.

2242 **4.10.2 Releasing a Human Task**

2243 The current actual owner of a human task can *release* a task to again make it available for all potential
2244 owners. A task can be released from active states that have an actual owner (*Reserved*, *InProgress*),
2245 transitioning it into the *Ready* state. Business data associated with the task (intermediate result data, ad-
2246 hoc attachments and comments) is kept.

2247 A task that is currently *InProgress* can be stopped by the actual owner, transitioning it into state
2248 *Reserved*. Business data associated with the task as well as its actual owner is kept.

2249 **4.10.3 Delegating or Forwarding a Human Task**

2250 Task's potential owners, actual owner or business administrator can *delegate* a task to another user,
2251 making that user the actual owner of the task, and also adding her to the list of potential owners in case
2252 she is not, yet. A task can be delegated when it is in an active state (*Ready*, *Reserved*, *InProgress*), and
2253 transitions the task into the *Reserved* state. Business data associated with the task is kept.

2254 Similarly, task's potential owners, actual owner or business administrator can forward an active task to
2255 another person or a set of people, replacing himself by those people in the list of potential owners.
2256 Potential owners can only forward tasks that are in the *Ready* state. Forwarding is possible if the task has
2257 a set of individually assigned potential owners, not if its potential owners are assigned using one or many
2258 groups. If the task is in the *Reserved* or *InProgress* state then the task is implicitly released first, that is,
2259 the task is transitioned into the *Ready* state. Business data associated with the task is kept. The user
2260 performing the forward is removed from the set of potential owners of the task, and the forwarder is
2261 added to the set of potential owners.

2262 **4.10.4 Sub Task Event Propagation**

2263 Task state transitions may be caused by the invocation of API operations (see section 7 "Programming
2264 Interfaces") or by events (see section 8 "Interoperable Protocol for Advanced Interaction with Human
2265 Tasks").

2266 If a task has sub tasks then some state transitions are propagated to these sub tasks. Conversely, if a
 2267 task has a parent task then some state transitions are propagated to that parent task.

2268 The following table defines how task state transitions MUST be propagated to sub tasks and to parent
 2269 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”)

2270 All other task state transitions MUST NOT affect sub tasks or a parent task.

2271 **4.11 History of a Human Task**

2272 Task lifecycle state changes and data changes are maintained as a history of task events. Task events
 2273 contain the following data:

2274 **Task Event**

- 2275 • event id
- 2276 • event time
- 2277 • task id
- 2278 • user (principal) that caused the state change
- 2279 • event type (e.g. claim task).
- 2280 • event data (e.g. data used in setOutput) and fault name (event was setFault)
- 2281 • startOwner - the actual owner before the event.
- 2282 • endOwner - the actual owner after the event.
- 2283 • task status at the end of the event

2284 For example, if the User1 delegated a task to User2, then the user and startOwner would be User1,
 2285 endOwner would be User2. The event data would be the <htt:organizationalEntity/> element used in the
 2286 WSHT delegate operation.

2287 The system generated attribute 'event id' MUST be unique on a per task basis.

2288 **4.11.1 Task Event Types and Data**

2289 Some task events (e.g. setOutput) may have data associated with event and others may not (e.g. claim).
 2290 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			

2291 4.11.2 Retrieving the History

2292 There is a `getTaskHistory` operation that allows a client to query the system and retrieve a list of task
2293 events that represent the history of the task. This operation can:

- 2294 • Return a list of task events with optional data
- 2295 • Return a list of task events without optional event data
- 2296 • Return a subset of the events based on a range (for paging)
- 2297 • Return a filtered list of events.

2298 The option to whether or not to include event data is useful since in some cases the event data content
2299 (e.g. `setOutput`) may be large. In a typical case, an API client should be able to query the system to get a
2300 "light weight" response of events (e.g. with out event data) and then when necessary, make an additional
2301 API call to get a specific event details with data. The latter can be accomplished by specifying the event id
2302 when invoking the `getTaskHistory` operation.

2303 The XML Schema definition of the filter is the following:

```

2304 <xsd:complexType name="tTaskHistoryFilter">
2305   <xsd:choice>
2306     <xsd:element name="eventId" type="xsd:integer" />
2307     <!-- Filter to allow narrow down query by status, principal,
2308          event Type. -->
2309     <xsd:sequence>
2310       <xsd:element name="status" type="tStatus" minOccurs="0"
2311                   maxOccurs="unbounded" />
2312       <xsd:element name="eventType" type="tTaskEventType" minOccurs="0"
2313                   maxOccurs="unbounded" />
2314       <xsd:element name="principal" type="xsd:string" minOccurs="0" />
2315       <xsd:element name="afterEventTime" type="xsd:dateTime"
2316                   minOccurs="0" />
2317       <xsd:element name="beforeEventTime" type="xsd:dateTime"
2318                   minOccurs="0" />
2319     </xsd:sequence>
2320   </xsd:choice>
2321 </xsd:complexType>
2322
2323 <xsd:simpleType name="tTaskEventType">
2324   <xsd:restriction base="xsd:string">
2325     <xsd:enumeration value="create" />
2326     <xsd:enumeration value="claim" />
2327     <xsd:enumeration value="start" />
2328     <xsd:enumeration value="stop" />
2329     <xsd:enumeration value="release" />
2330     <xsd:enumeration value="suspend" />

```

```

2331 <xsd:enumeration value="suspendUntil" />
2332 <xsd:enumeration value="resume" />
2333 <xsd:enumeration value="complete" />
2334 <xsd:enumeration value="remove" />
2335 <xsd:enumeration value="fail" />
2336 <xsd:enumeration value="setPriority" />
2337 <xsd:enumeration value="addAttachment" />
2338 <xsd:enumeration value="deleteAttachment" />
2339 <xsd:enumeration value="addComment" />
2340 <xsd:enumeration value="updateComment" />
2341 <xsd:enumeration value="deleteComment" />
2342 <xsd:enumeration value="skip" />
2343 <xsd:enumeration value="forward" />
2344 <xsd:enumeration value="delegate" />
2345 <xsd:enumeration value="setOutput" />
2346 <xsd:enumeration value="deleteOutput" />
2347 <xsd:enumeration value="setFault" />
2348 <xsd:enumeration value="deleteFault" />
2349 <xsd:enumeration value="activate" />
2350 <xsd:enumeration value="nominate" />
2351 <xsd:enumeration value="setGenericHumanRole" />
2352 <xsd:enumeration value="expire" />
2353 <xsd:enumeration value="escalated" />
2354 </xsd:restriction>
2355 </xsd:simpleType>

```

2356 The XML Schema definition of events returned for the history is the following:

```

2357 <xsd:element name="taskEvent">
2358   <xsd:complexType>
2359     <xsd:annotation>
2360       <xsd:documentation>
2361         A detailed event that represents a change in the task's state.
2362       </xsd:documentation>
2363     </xsd:annotation>
2364     <xsd:sequence>
2365       <!-- event id - unique per task -->
2366       <xsd:element name="id" type="xsd:integer" />
2367       <!-- event date time -->
2368       <xsd:element name="eventTime" type="xsd:dateTime" />
2369       <!-- task ID -->
2370       <xsd:element name="identifier" type="xsd:anyURI" />
2371       <xsd:element name="principal" type="xsd:string" minOccurs="0"
2372         nillable="true" />
2373       <!-- Event type. Note - using a restricted type limits
2374         extensibility to add custom event types. -->
2375       <xsd:element name="eventType" type="tTaskEventType" />
2376       <!-- actual owner of the task before the event -->
2377       <xsd:element name="startOwner" type="xsd:string" minOccurs="0"
2378         nillable="true" />
2379       <!-- actual owner of the task after the event -->
2380       <xsd:element name="endOwner" type="xsd:string" minOccurs="0"
2381         nillable="true" />
2382       <!-- WSHT task status -->
2383       <xsd:element name="status" type="tStatus" />
2384       <!-- boolean to indicate this event has optional data -->
2385       <xsd:element name="hasData" type="xsd:boolean" minOccurs="0" />
2386       <xsd:element name="eventData" type="xsd:anyType" minOccurs="0"

```

```
2387         nillable="true" />
2388     <xsd:element name="faultName" type="xsd:string" minOccurs="0"
2389         nillable="true" />
2390     <!-- extensibility -->
2391     <xsd:any namespace="##other" processContents="lax" minOccurs="0"
2392         maxOccurs="unbounded" />
2393 </xsd:sequence>
2394 </xsd:complexType>
2395 </xsd:element>
2396
```

2397 5 Lean Tasks

2398 The <leanTask> element is used to specify human tasks. This section introduces the syntax for the
2399 element, and individual properties are explained in subsequent sections.

2400 5.1 Overall Syntax

2401 The element <leanTask> derives from the type htd:tTask, with the following augmentations:

```
2402 <htd:leanTask>  
2403   <htd:interface>...</htd:interface>  
2404   <htd:messageSchema>...</htd:messageSchema>  
2405   ... All elements from htd:task except <interface> and <composition> ...  
2406   <htd:composition>...</htd:composition>  
2407 </htd:leanTask>
```

2408 5.2 Properties

2409 The following attributes and elements are defined for lean tasks and are different from the definition of
2410 htd:task:

- 2411 • `interface` – Lean tasks are created through the CreateLeanTask operation (section 7.3.4), and
2412 their input message is derived from the messageSchema element. Therefore, an interface
2413 element might contradict that information, and to prevent that, interface is banned.
- 2414 • `messageSchema` – Identifies the schema of the inputMessage and outputMessage for the lean
2415 task, and if the renderings element is not defined, the WS-HumanTask Processor can use this to
2416 generate a rendering or pass this data directly to a WS-HumanTask Client such that the
2417 rendering is generated from the messageSchema.
- 2418 • `composition` – Lean tasks cannot have explicitly declared subtasks as defined for composite
2419 tasks (section 4.6), consequently, this element is banned.

2420 5.3 Message Schema

2421 This element references the schema of the data that is used for both the input and output messages of
2422 the lean task.

```
2423 <messageSchema>  
2424   <messageField name="xsd:NCName" type="xsd:QName">*  
2425     <messageDisplay xml:lang="xsd:language"?>+  
2426     Language specific display  
2427   </messageDisplay>  
2428   <messageChoice namevalue="xsd:NCNameanySimpleType">*  
2429     <messageDisplay xml:lang="xsd:language"?>+  
2430     Language specific display  
2431   </messageDisplay>  
2432   </messageChoice>  
2433   </messageField>  
2434 </messageSchema>
```

2435 The <messageSchema> element specifies the data that a Lean Task accepts. As it is currently defined, a
2436 WS-HumanTask Processor could render the following form elements in a way that only requires vendor-
2437 specific knowledge between the WS-HumanTask Processor and the WS-HumanTask Client and no
2438 vender-specific knowledge between the WS-HumanTask Processor and the WS-HumanTask Parent:

- 2439 • String
- 2440 • Integer

- 2441 • Float
- 2442 • Date Time
- 2443 • Bool
- 2444 • Enumeration (Choice)

2445 Each of these is accomplished by using an instance of a <messageField>. For string, integer, float,
 2446 datetime, and boolean fields, this is accomplished by using the type attribute of the <messageField>.
 2447 The supported set of values are: xsd:string, xsd:integer, xsd:float, xsd:datetime, and
 2448 xsd:boolean, all respectively matching the list above. If a simple rendering language like HTML were
 2449 used, this could be accomplished by using a textbox control that simply had special rules about the format
 2450 of its input.

2451 The enumeration field uses a combination of one <messageField> element and possibly many child
 2452 <messageChoice> elements. Each child <messageChoice> represents one possible option that could
 2453 be selected from the enumeration. If a simple rendering language like HTML were used, this could be
 2454 shown using radio buttons, a dropdown list, or a listbox that only supports single selection.

2455 For all <messageField> and <messageChoice> elements, it is possible to specify a per-language
 2456 <messageDisplay> element. It uses xml:lang, a standard XML attribute, to define the language of the
 2457 enclosed information. This attribute uses tags according to RFC 1766 (see [RFC1766]). There could be
 2458 zero or more <messageDisplay> elements. A <messageField> or <messageChoice> MUST NOT
 2459 specify multiple <messageDisplay> elements having the same value for the attribute xml:lang.

2460 The combination of <messageSchema> and <possibleOutcomes> can be used to generate a form of
 2461 sufficient functionality for many simple tasks, precluding the need for a renderings element.

2462 **Example:**

```

2463 <messageSchema>
2464   <messageField name="amount" type="xsd:float">
2465     <messageDisplay xml:lang="en-us">Amount</messageDisplay>
2466     <messageDisplay xml:lang="fr-fr">Quantité</messageDisplay>
2467   </messageField>
2468   <messageField name="currencyUnit" type="xsd:string">
2469     <messageDisplay xml:lang="en-us">Currency</messageDisplay>
2470     <messageDisplay xml:lang="fr-fr">Devise</messageDisplay>
2471     <messageChoice namevalue="USD">
2472       <messageDisplay xml:lang="en-us">US Dollars</messageDisplay>
2473       <messageDisplay xml:lang="fr-fr">US Dollars</messageDisplay>
2474     </messageChoice>
2475     <messageChoice namevalue="EURO">
2476       <messageDisplay xml:lang="en-us">Euro Dollars</messageDisplay>
2477       <messageDisplay xml:lang="fr-fr">Euros</messageDisplay>
2478     </messageChoice>
2479   </messageField>
2480 </messageSchema>
  
```

2481

2482

2483 **5.4 Example: ToDoTask**

2484 The following XML could be used for a simple 'ToDoTask':

```
2485 <htd:task name="ToDoTask">
2486   <htd:messageSchema />
2487   <htd:possibleOutcomes>
2488     <htd:possibleOutcome name="Completed" />
2489     ... language specific translations ...
2490   </htd:possibleOutcomes>
2491   <htd:delegation potentialDelegates="anybody" />
2492   <htd:presentationElements>
2493     <htd:name>ToDo Task</htd:name>
2494     ... language specific translations ...
2495     <htd:subject>Please complete the described work</htd:subject>
2496     ... language specific translations ...
2497     <htd:description contentType="mimeTypeString" />
2498     ... language specific translations ...
2499   </htd:presentationElements>
2500 </htd:task>
```

2501

6 Notifications

2502 Notifications are used to notify a person or a group of people of a noteworthy business event, such as
2503 that a particular order has been approved, or a particular product is about to be shipped. They are also
2504 used in escalation actions to notify a user that a task is overdue or a task has not been started yet. The
2505 person or people to whom the notification will be assigned to could be provided, for example, as result of
2506 a people query to organizational model.

2507 Notifications are simple human interactions that do not block the progress of the caller, that is, the caller
2508 does not wait for the notification to be completed. Moreover, the caller cannot influence the execution of
2509 notifications, e.g. notifications are not terminated if the caller terminates. The caller, i.e. an application, a
2510 business process or an escalation action, initiates a notification passing the required notification data. The
2511 notification appears on the task list of all notification recipients. After a notification recipient removes it,
2512 the notification disappears from the recipient's task list.

2513 A notification MAY have multiple recipients and optionally one or many business administrators. The
2514 generic human roles task initiator, task stakeholders, potential owners, actual owner and excluded
2515 owners play no role.

2516 Presentation elements and task rendering, as described in sections 4.3 and 4.4 respectively, are used for
2517 notifications also. In most cases the subject line and description are sufficient information for the
2518 recipients, especially if the notifications are received in an e-mail client or mobile device. But in some
2519 cases the notifications can be received in a proprietary client so the notification can support a proprietary
2520 rendering format to enable this to be utilized to the full, such as for rendering data associated with the
2521 caller invoking the notification. For example, the description could include a link to the process audit trail
2522 or a button to navigate to business transactions involved in the underlying process.

2523 Notifications do not have ad-hoc attachments, comments or deadlines.

2524 6.1 Overall Syntax

2525 Definition of notifications

```
2526 <htd:notification name="NCName" >  
2527  
2528   <htd:interface portType="QName" operation="NCName" />  
2529  
2530   <htd:priority expressionLanguage="anyURI"??>  
2531     integer-expression  
2532   </htd:priority>  
2533  
2534   <htd:peopleAssignments>  
2535  
2536     <htd:recipients>  
2537       ...  
2538     </htd:recipients>  
2539  
2540     <htd:businessAdministrators?>  
2541       ...  
2542     </htd:businessAdministrators>  
2543  
2544   </htd:peopleAssignments>  
2545  
2546   <htd:presentationElements>  
2547     ...  
2548   </htd:presentationElements>  
2549  
2550   <htd:renderings?>
```



```
2551     ...
2552     </htd:renderings>
2553
2554 </htd:notification>
```

2555 6.2 Properties

2556 The following attributes and elements are defined for notifications:

- 2557 • `name`: This attribute is used to specify the name of the notification. The name combined with the
2558 target namespace MUST uniquely identify a notification in the notification definition. The attribute
2559 is mandatory. It is not used for notification rendering.
- 2560 • `interface`: This element is used to specify the operation used to invoke the notification. The
2561 operation is specified using WSDL, that is a WSDL port type and WSDL operation are defined.
2562 The element and its `portType` and `operation` attributes are mandatory. In the `operation`
2563 attribute, a WS-HumanTask Definition MUST reference a one-way WSDL operation.
- 2564 • `priority`: This element is used to specify the priority of the notification. It is an optional
2565 element which value is an integer expression. If present then the WS-HumanTask Definition
2566 MUST specify a value between 0 and 10, where 0 is the highest priority and 10 is the lowest. If
2567 not present, the priority of the notification is considered as 5. The result of the expression
2568 evaluation is of type `htt:tPriority`. The `expressionLanguage` attribute specifies the
2569 language used in the expression. The attribute is optional. If not specified, the default language
2570 as inherited from the closest enclosing element that specifies the attribute is used.
- 2571 • `peopleAssignments`: This element is used to specify people assigned to the notification. The
2572 element is mandatory. A WS-HumanTask Definition MUST include a people assignment for
2573 recipients and MAY include a people assignment for business administrators.
- 2574 • `presentationElements`: The element is used to specify different information used to display
2575 the notification, such as name, subject and description, in a task list. The element is mandatory.
2576 See section 4.3 for more information on presentation elements.
- 2577 • `rendering`: The element is used to specify rendering method. It is optional. If not present,
2578 notification rendering is implementation dependent. See section 4.4 for more information on
2579 rendering.

2580 6.3 Notification Behavior and State Transitions

2581 Same as human tasks, notifications are in pseudo-state *Inactive* before they are activated. Once they are
2582 activated they move to the *Ready* state. This state is observable, that is, when querying for notifications
2583 then all notifications in state *Ready* are returned. When a notification is removed then it moves into the
2584 final pseudo-state *Removed*.

2585 7 Programming Interfaces

2586 7.1 Operations for Client Applications

2587 A number of applications are involved in the life cycle of a task. These comprise:

- 2588 • The task list client, i.e. a client capable of displaying information about the task under
2589 consideration
- 2590 • The requesting application, i.e. any partner that has initiated the task
- 2591 • The supporting application, i.e. an application launched by the task list client to support
2592 processing of the task.

2593 The task infrastructure provides access to a given task. It is important to understand that what is meant
2594 by *task list client* is the software that presents a UI to one authenticated user, irrespective of whether this
2595 UI is rendered by software running on server hardware (such as in a portals environment) or client
2596 software (such as a client program running on a users workstation or PC).

2597 A given task exposes a set of operations to this end. A WS-HumanTask Processor MUST provide the
2598 operations listed below and an application (such as a task list client) can use these operations to
2599 manipulate the task. All operations MUST be executed in a synchronous fashion and MUST return a fault
2600 if certain preconditions do not hold. For operations that are not expected to return a response they MAY
2601 return a void message. The above applies to notifications also.

2602 An operation takes a well-defined set of parameters as its input. Passing an illegal parameter or an illegal
2603 number of parameters MUST result in the `hta:illegalArgumentFault` being returned. Invoking an
2604 operation that is not allowed in the current state of the task MUST result in an
2605 `hta:illegalStateFault`.

2606 By default, the identity of the person on behalf of which the operation is invoked is passed to the task.
2607 When the person is not authorized to perform the operation the `hta:illegalAccessFault` and
2608 `hta:recipientNotAllowed` MUST be returned in the case of tasks and notifications respectively.

2609 Invoking an operation that does not apply to the task type (e.g., invoking claim on a notification) MUST
2610 result in an `hta:illegalOperationFault`.

2611 The language of the person on behalf of which the operation is invoked is assumed to be available to
2612 operations requiring that information, e.g., when accessing presentation elements.

2613 For an overview of which operations are allowed in what state, refer to section 4.10 "Human Task
2614 Behavior and State Transitions". For a formal definition of the allowed operations, see Appendix D "WS-
2615 HumanTask Client API Port Type".

2616 For information which generic human roles are authorized to perform which operations, refer to section
2617 7.1.4 "Operation Authorizations".

2618 This specification does not stipulate the authentication, language passing, addressing, and binding
2619 scheme employed when calling an operation. This can be achieved using different mechanisms (e.g. WS-
2620 Security, WS-Addressing).

2621 7.1.1 Participant Operations

2622 Operations are executed by end users, i.e. actual or potential owners. The identity of the user is implicitly
2623 passed when invoking any of the operations listed in the table below.

2624 If the task is in a predefined state listed as valid pre-state before the operation is invoked then, upon
2625 successful completion, the task MUST be in the post state defined for the operation. If the task is in a
2626 predefined state that is not listed as valid pre-state before the operation is invoked then the operation
2627 MUST be rejected and MUST NOT cause a task state transition.

2628 All of the operations below apply to tasks and sub tasks only unless specifically noted below.

2629 The column “**Supports Batch Processing**” below indicates if an operation can be used to process
 2630 multiple human tasks at the same time. One or more operations on individual tasks may fail without
 2631 causing the overall batch operation to fail.
 2632

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 (http:priority) 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)
addComment getAttachmentInfos	Get attachment information for all attachments associated with the task. 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"> • list of attachment data (list of http:attachmentInfo) comment identifier 	No	(any state)	(no state transition)
claim getAttachment	GetClaim responsibility for a task, i.e. set the task attachment with the given identifier to status <i>Reserved</i>	In <ul style="list-style-type: none"> • task identifier • attachment identifier Out <ul style="list-style-type: none"> • http:attachmentInfo • void 	No Yes	(any state) Ready	(no state transition) Reserved

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
complete	Execution of the task finished successfully. The fault <code>hta:illegalState</code> MUST be returned if the task interface defines non-empty task output but no output data is provided as the input parameter and the task output data has not been set previously, e.g. using operation <code>setOutput</code>.	<u>In</u> <ul style="list-style-type: none"> task identifier output data of task (optional) <u>Out</u> <ul style="list-style-type: none"> void 	Yes	InProgress	Completed
delegate	Assign the task to one user and set the task to state <code>Reserved</code>. 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.	<u>In</u> <ul style="list-style-type: none"> task identifier organizational entity (<code>htt:tOrganizationalEntity</code>) <u>Out</u> <ul style="list-style-type: none"> void 	Yes	Ready Reserved InProgress	Reserved
<code>deleteAttachment</code>	Delete the attachment with the specified identifier from the task. Attachments provided by the enclosing context MUST NOT be affected by this operation.	<u>In</u> <ul style="list-style-type: none"> task identifier attachment identifier <u>Out</u> <ul style="list-style-type: none"> void 	No	(any state)	(no state transition)
deleteComment addComment	Add a comment to a task. Returns an identifier that can be used to later update or delete the comment. Deletes the identified comment.	<u>In</u> <ul style="list-style-type: none"> task identifier plain text <u>Out</u> <ul style="list-style-type: none"> comment identifier <u>Out</u> <ul style="list-style-type: none"> void 	No	(any state)	(no state transition)

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
deleteFaultupdateComment	Updates Deletes the identified comment with fault name and fault data of the supplied new text.task.	In <ul style="list-style-type: none"> task identifier comment identifier plain text Out <ul style="list-style-type: none"> void 	No	(any state) <u>InProgress</u>	(no state transition)
deleteOutputdeleteComment	Deletes the identified comment.output data of the task.	In <ul style="list-style-type: none"> task identifier comment identifier Out <ul style="list-style-type: none"> void 	No	(any state) <u>InProgress</u>	(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 http:comment) 	No	(any state)	(no state transition)
failskip	Skip <u>Execution of the task- fails and a fault is returned.</u> If the task is not skipable then the The <u>fault</u> hta:illegalOperationFault MUST be returned if the task interface defines no faults. The fault <u>hta:illegalStateFault MUST be returned if the task interface defines at least one faults but either fault name or fault data is not provided and it has not been set previously, e.g. using operation setFault.</u>	In <ul style="list-style-type: none"> task identifier <u>fault (optional) – contains the fault name and fault data</u> Out <ul style="list-style-type: none"> void 	Yes	Created Ready Reserved <u>InProgress</u>	Obsolete <u>Failed</u>

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:taskIdentifier</code>) Out <ul style="list-style-type: none"> void 	Yes	Ready Reserved InProgress	Ready
getAttachmentDelegate	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. Get the task attachment with the given identifier.	In <ul style="list-style-type: none"> task identifier organizational entity (<code>http:taskIdentifier</code>) Out <ul style="list-style-type: none"> void attachment identifier Out <ul style="list-style-type: none"> http:attachment 	YesNo	Ready Reserved InProgress(any state)	Reserved(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>http:attachmentInfo</code>) 	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>http:comment</code>) 	No	(any state)	(no state transition)

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

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
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)
getSubtaskIdentifiers getTaskDetails	Applies to both tasks and notifications. Returns a data object the identifiers of type <u>http://tTaskDetails</u> all already created sub tasks of a task	In <ul style="list-style-type: none"> task identifier Out <ul style="list-style-type: none"> <u>list of task (http://tTaskDetails) identifiers</u> 	No	(any state)	(no state transition)
getSubtasks	Returns all sub tasks of a task (created instances + not yet created sub task definitions)	In <ul style="list-style-type: none"> <u>task identifier</u> Out <ul style="list-style-type: none"> <u>list of tasks (list of http://tTask)</u> 	<u>No</u>	<u>(any state)</u>	<u>(no state transition)</u>
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)
getTaskDetails getTaskOperations	Applies to <u>both</u> tasks and notifications. Returns <u>list a data object of operations that are available to the authorized user given the user's role and the state of the task.type</u> <u>http://tTaskDetails</u>	In <ul style="list-style-type: none"> task identifier Out <ul style="list-style-type: none"> <u>List of available operation.task (http://tTaskDetails)</u> 	No	(any state)	(no state transition)

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
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)
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)

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
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)
getTaskHistory	Get a list of events representing the history of the task. <i>Filter</i> allows narrowing the results by status, principal, event Type. <i>startIndex</i> and <i>maxTasks</i> are integers that allow paging of the results. <i>includeData</i> is a Boolean. Data is included with the returned events only if this is true.	In <ul style="list-style-type: none"> • task identifier • filter (htt:tTaskHistoryFilter) • startIndex • maxTasks • includeData Out <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
getTaskInstance Data	<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
getTaskOperations getSubtasks	Returns all sub tasks of a task (created instances + not yet created sub task definitions) 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 tasks (list of http:Task) available operation. 	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"> http:Task 	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 <code>hta:illegalArgumentFault</code> 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)
setTaskStartDeadlineExpressionisSubtask	<p>SetsReturns true if a deadline-expression for the named start deadline task is a subtask of the superior composite task</p>	<p>In</p> <ul style="list-style-type: none"> task identifier deadline name deadline expression <p>Out</p> <ul style="list-style-type: none"> voidboolean 	YesNo	Created Ready Reserved In Progress (any state)	(no state transition)
release	<p>Release the task, i.e. set the task back to status Ready.</p>	<p>In</p> <ul style="list-style-type: none"> task identifier <p>Out</p> <ul style="list-style-type: none"> void 	Yes	InProgress Reserved	Ready
setTaskStartDurationExpressionremove	<p>Sets a duration expression for the named start deadline of the task. 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.</p>	<p>In</p> <ul style="list-style-type: none"> task identifier deadline name duration expression <p>Out</p> <ul style="list-style-type: none"> void 	Yes	Created Ready Reserved In-Progress (Notification state)	(noRemoved (Notification state transition))

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
<u>resume</u>	<u>Resume a suspended task.</u>	<u>In</u> <ul style="list-style-type: none"> <u>task identifier</u> <u>Out</u> <ul style="list-style-type: none"> <u>void</u> 	<u>Yes</u>	<u>Suspended/Ready</u> <u>Suspended/Reserved</u> <u>Suspended/InProgress</u>	<u>Ready (from Suspended/Ready)</u> <u>Reserved (from Suspended/Reserved)</u> <u>InProgress (from Suspended/InProgress)</u>
<u>setFault</u>	<u>Set the fault data of the task.</u> <u>The fault hta:illegalOperationFault MUST be returned if the task interface defines no faults.</u>	<u>In</u> <ul style="list-style-type: none"> <u>task identifier</u> <u>fault – contains the fault name and fault data</u> <u>Out</u> <ul style="list-style-type: none"> <u>void</u> 	<u>No</u>	<u>InProgress</u>	<u>(no state transition)</u>
<u>setOutput</u>	<u>Set the data for the part of the task's output message.</u>	<u>In</u> <ul style="list-style-type: none"> <u>task identifier</u> <u>part name (optional for single part messages)</u> <u>output data of task</u> <u>Out</u> <ul style="list-style-type: none"> <u>void</u> 	<u>No</u>	<u>InProgress</u>	<u>(no state transition)</u>
<u>setPriority</u>	<u>Change the priority of the task. The WS-HumanTask Client MUST specify the integer value of the new priority.</u>	<u>In</u> <ul style="list-style-type: none"> <u>task identifier</u> <u>priority (http:tpriority)</u> <u>Out</u> <ul style="list-style-type: none"> <u>void</u> 	<u>Yes</u>	<u>(any state)</u>	<u>(no state transition)</u>

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
setTaskCompletionDeadlineExpression	Sets a deadline expression for the named completion deadline of the task	In <ul style="list-style-type: none"> task identifier deadline name deadline expression Out void	Yes	Created Ready Reserved In Progress	(no state transition)
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)
<u>setTaskStartDeadlineExpression</u>	<u>Sets a deadline expression for the named start deadline of the task</u>	<u>In</u> <ul style="list-style-type: none"> <u>task identifier</u> <u>deadline name</u> <u>deadline expression</u> <u>Out</u> <ul style="list-style-type: none"> <u>void</u> 	<u>Yes</u>	<u>Created</u> <u>Ready</u> <u>Reserved</u> <u>In Progress</u>	<u>(no state transition)</u>
<u>setTaskStartDurationExpression</u>	<u>Sets a duration expression for the named start deadline of the task</u>	<u>In</u> <ul style="list-style-type: none"> <u>task identifier</u> <u>deadline name</u> <u>duration expression</u> <u>Out</u> <u>void</u>	<u>Yes</u>	<u>Created</u> <u>Ready</u> <u>Reserved</u> <u>In Progress</u>	<u>(no state transition)</u>
<u>skip</u>	<u>Skip the task.</u> <u>If the task is not skipable then the fault <code>hta:illegalOperationFault</code> MUST be returned.</u>	<u>In</u> <ul style="list-style-type: none"> <u>task identifier</u> <u>Out</u> <ul style="list-style-type: none"> <u>void</u> 	<u>Yes</u>	<u>Created</u> <u>Ready</u> <u>Reserved</u> <u>InProgress</u>	<u>Obsolete</u>

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
<u>start</u>	<u>Start the execution of the task, i.e. set the task to status <i>InProgress</i>.</u>	<u>In</u> <ul style="list-style-type: none"> • <u>task identifier</u> <u>Out</u> <ul style="list-style-type: none"> • <u>void</u> 	<u>Yes</u>	<u>Ready</u> <u>Reserved</u>	<u>InProgress</u>
<u>stop</u>	<u>Cancel/stop the processing of the task. The task returns to the <i>Reserved</i> state.</u>	<u>In</u> <ul style="list-style-type: none"> • <u>task identifier</u> <u>Out</u> <ul style="list-style-type: none"> • <u>void</u> 	<u>Yes</u>	<u>InProgress</u>	<u>Reserved</u>
<u>suspend</u>	<u>Suspend the task.</u>	<u>In</u> <ul style="list-style-type: none"> • <u>task identifier</u> <u>Out</u> <ul style="list-style-type: none"> • <u>void</u> 	<u>Yes</u>	<u>Ready</u> <u>Reserved</u> <u>InProgress</u>	<u>Suspended/Ready (from Ready)</u> <u>Suspended/Reserved (from Reserved)</u> <u>Suspended/InProgress (from InProgress)</u>
<u>suspendUntil</u>	<u>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.</u>	<u>In</u> <ul style="list-style-type: none"> • <u>task identifier</u> • <u>time period</u> • <u>point of time</u> <u>Out</u> <ul style="list-style-type: none"> • <u>void</u> 	<u>Yes</u>	<u>Ready</u> <u>Reserved</u> <u>InProgress</u>	<u>Suspended/Ready (from Ready)</u> <u>Suspended/Reserved (from Reserved)</u> <u>Suspended/InProgress (from InProgress)</u>
<u>updateComment</u>	<u>Updates the identified comment with the supplied new text.</u>	<u>In</u> <ul style="list-style-type: none"> • <u>task identifier</u> • <u>comment identifier</u> • <u>plain text</u> <u>Out</u> <ul style="list-style-type: none"> • <u>void</u> 	<u>No</u>	<u>(any state)</u>	<u>(no state transition)</u>

2633

2634 7.1.2 Simple Query Operations

2635 Simple query operations allow retrieving task data. These operations MUST be supported by a WS-
2636 HumanTask Processor. The identity of the user is implicitly passed when invoking any of the following
2637 operations.

2638 The following operations will return both matching tasks and sub tasks.
 2639

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 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</p>	<p>In</p> <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 <p>Out</p> <ul style="list-style-type: none"> • list of tasks (list of <code>htt:tTaskAbstract</code>) 	Any

Operation Name	Description	Parameters	Authorization
	<p>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 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</p>	<p>In</p> <ul style="list-style-type: none"> • task type ("ALL" "TASKS" "NOTIFICATIONS") • generic human role • work queue • status list • where clause • created-on clause • maxTasks <p>Out</p> <ul style="list-style-type: none"> • list of tasks (list of <code>htt:tTaskDetails</code>) 	Any

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

2640

2641 The return types `tTaskAbstract` and `tTaskDetails` are defined in section 3.8.4 “Data Types for Task
 2642 Instance Data”.

2643 **Simple Task View**

2644 The table below lists the task attributes available to the simple query operations. This view is used when
 2645 defining the where clause of any of the above query operations.

2646

Column Name	Type
ID	xsd:string

Column Name	Type
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
SearchBy	xsd:string
Outcome	xsd:string

2647

2648 7.1.3 Advanced Query Operation

2649 The advanced query operation is used by the task list client to perform queries not covered by the simple
 2650 query operations defined in 7.1.2. A WS-HumanTask Processor MAY support this operation. An
 2651 implementation MAY restrict the results according to authorization of the invoking user.

2652

2653 The following operations will return both matching tasks and sub tasks.

2654

Operation Name	Description	Parameters
query	Retrieve task data. All clauses assume a (pseudo-) SQL syntax. If	In <ul style="list-style-type: none"> select clause

	<p>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.</p>	<ul style="list-style-type: none"> • where clause • order-by clause • maxTasks • taskIndexOffset <p>Out</p> <ul style="list-style-type: none"> • task query result set (htt:tTaskQueryResultSet)
--	---	---

2655

2656 **ResultSet Data Type**

2657 This is the result set element that is returned by the query operation.

```

2658 <xsd:element name="taskQueryResultSet" type="tTaskQueryResultSet" />
2659 <xsd:complexType name="tTaskQueryResultSet">
2660   <xsd:sequence>
2661     <xsd:element name="row" type="tTaskQueryResultRow"
2662               minOccurs="0" maxOccurs="unbounded" />
2663   </xsd:sequence>
2664 </xsd:complexType>
2665

```

2666 The following is the type of the row element contained in the result set. The value in the row are returned
2667 in the same order as specified in the select clause of the query.

```

2668 <xsd:complexType name="tTaskQueryResultRow">
2669   <xsd:choice minOccurs="0" maxOccurs="unbounded">
2670     <xsd:element name="id" type="xsd:stringanyURI"/>
2671     <xsd:element name="taskType" type="xsd:string"/>
2672     <xsd:element name="name" type="xsd:QName"/>
2673     <xsd:element name="status" type="tStatus"/>
2674     <xsd:element name="priority" type="htt:tPriority"/>
2675     <xsd:element name="taskInitiator"
2676               type="htt:tUser"/>
2677     <xsd:element name="taskStakeholders"
2678               type="htt:tOrganizationalEntity"/>
2679     <xsd:element name="potentialOwners"
2680               type="htt:tOrganizationalEntity"/>
2681     <xsd:element name="businessAdministrators"
2682               type="htt:tOrganizationalEntity"/>
2683     <xsd:element name="actualOwner" type="htt:tUser"/>
2684     <xsd:element name="notificationRecipients"
2685               type="htt:tOrganizationalEntity"/>
2686     <xsd:element name="createdTime" type="xsd:dateTime"/>
2687     <xsd:element name="createdBy" type="xsd:string"/>
2688     <xsd:element name="lastModifiedTime" type="xsd:dateTime"/>
2689     <xsd:element name="lastModifiedBy" type="xsd:string"/>
2690     <xsd:element name="activationTime" type="xsd:dateTime"/>
2691     <xsd:element name="expirationTime" type="xsd:dateTime"/>
2692     <xsd:element name="isSkipable" type="xsd:boolean"/>
2693     <xsd:element name="hasPotentialOwners" type="xsd:boolean"/>
2694     <xsd:element name="startByTime" type="xsd:dateTime"/>
2695     <xsd:element name="completeByTime" type="xsd:dateTime"/>
2696     <xsd:element name="presentationName" type="tPresentationName"/>
2697     <xsd:element name="presentationSubject"
2698               type="tPresentationSubject"/>

```

```
2699 <xsd:element name="renderingMethodName" type="xsd:QName"/>
2700 <xsd:element name="hasOutput" type="xsd:boolean"/>
2701 <xsd:element name="hasFault" type="xsd:boolean"/>
2702 <xsd:element name="hasAttachments" type="xsd:boolean"/>
2703 <xsd:element name="hasComments" type="xsd:boolean"/>
2704 <xsd:element name="escalated" type="xsd:boolean"/>
2705 <xsd:element name="parentTaskId" type="xsd:stringanyURI"/>
2706 <xsd:element name="hasSubTasks" type="xsd:boolean"/>
2707 <xsd:element name="searchBy" type="xsd:string"/>
2708 <xsd:element name="outcome" type="xsd:string"/>
2709 <xsd:element name="taskOperations" type="tTaskOperations"/>
2710 <xsd:any namespace="##other" processContents="lax"/>
2711 </xsd:choice>
2712 </xsd:complexType>
```

2713 **Complete Task View**

2714 The table below is the set of columns used when defining select clause, where clause, and order-by
2715 clause of query operations. Conceptually, this set of columns defines a universal relation. As a result the
2716 query can be formulated without specifying a from clause. A WS-HumanTask Processor MAY extend this
2717 view by adding columns.

2718

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	

2720

2721 **7.1.4 Administrative Operations**

2722 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)

2723

2724

2725 **7.1.5 Operation Authorizations**

2726 The table below summarizes the required authorizations in terms of generic human roles to execute
 2727 participant, query and administrative operations. Thus, it is a precise definition of the generic human roles
 2728 as well. The sign plus ('+') means that the operation MUST be available for the generic human role. The
 2729 sign minus ('-') means that the operation MUST NOT be available for the generic human role. 'n/a'
 2730 indicates that the operation is not applicable and thus MUST NOT be available for the generic human
 2731 role. 'MAY' defines that vendor MAY chose to support the operation for the generic human role.

2732 If a person has multiple generic human roles on a human task or notification and she is allowed to
 2733 perform an operation in any of the roles then the invocation of the operation will not fail, otherwise
 2734 hta:illegalAccessFault and hta:recipientNotAllowed MUST be returned in the case of tasks
 2735 and notifications respectively. If a person is included in the list of excluded owners of a task then she
 2736 MUST NOT perform any of the operations.

2737 All batch operations (operations with a name prefix "batch") may be invoked by any caller; no specific
 2738 authorization is required. Missing authorizations for operations on individual tasks result in a report entry
 2739 in the batch operation's response message.

2740

Operation \ Role	Task Initiator	Task Stakeholders	Potential Owners	Actual Owner	Business Administrator	Notification Recipients
<u>activate</u>	<u>+</u>	<u>+</u>	<u>n/a</u>	<u>n/a</u>	<u>+</u>	<u>-</u>
<u>addAttachment</u>	<u>MAY</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>n/a</u>
<u>addComment</u>	<u>MAY</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>n/a</u>
<u>batch*</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>
<u>claim</u>	<u>-</u>	<u>MAY</u>	<u>+</u>	<u>n/a</u>	<u>MAY</u>	<u>n/a</u>
<u>complete</u>	<u>-</u>	<u>MAY</u>	<u>n/a</u>	<u>+</u>	<u>MAY</u>	<u>n/a</u>
<u>delegate</u>	<u>MAY</u>	<u>+</u>	<u>MAY</u>	<u>+</u>	<u>+</u>	<u>n/a</u>
<u>deleteAttachment</u>	<u>MAY</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>n/a</u>
<u>deleteComment</u>	<u>MAY</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>n/a</u>
<u>deleteFault</u>	<u>-</u>	<u>MAY</u>	<u>n/a</u>	<u>+</u>	<u>MAY</u>	<u>n/a</u>
<u>deleteOutput</u>	<u>-</u>	<u>MAY</u>	<u>n/a</u>	<u>+</u>	<u>MAY</u>	<u>n/a</u>
<u>fail</u>	<u>-</u>	<u>MAY</u>	<u>n/a</u>	<u>+</u>	<u>MAY</u>	<u>n/a</u>
<u>forward</u>	<u>MAY</u>	<u>+</u>	<u>MAY</u>	<u>+</u>	<u>+</u>	<u>n/a</u>
<u>getAttachment</u>	<u>MAY</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>n/a</u>
<u>getAttachmentInfos</u>	<u>MAY</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>n/a</u>
<u>getComments</u>	<u>MAY</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>n/a</u>
<u>getFault</u>	<u>+</u>	<u>+</u>	<u>MAY</u>	<u>+</u>	<u>+</u>	<u>n/a</u>
<u>getInput</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>n/a</u>
<u>getMyTaskAbstracts</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>
<u>getMyTaskDetails</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>
<u>getOutcome</u>	<u>+</u>	<u>+</u>	<u>MAY</u>	<u>+</u>	<u>+</u>	<u>n/a</u>
<u>getOutput</u>	<u>+</u>	<u>+</u>	<u>MAY</u>	<u>+</u>	<u>+</u>	<u>n/a</u>
<u>getParentTask</u>	<u>+</u>	<u>+</u>	<u>MAY</u>	<u>+</u>	<u>+</u>	<u>n/a</u>
<u>getParentTaskIdentifier</u>	<u>+</u>	<u>+</u>	<u>MAY</u>	<u>+</u>	<u>+</u>	<u>n/a</u>
<u>getRendering</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>
<u>getRenderingTypes</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>
<u>getSubtaskIdentifiers</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>n/a</u>
<u>getSubtasks</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>n/a</u>
<u>getTaskDescription</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>
<u>getTaskDetails</u>	<u>MAY</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>
<u>getTaskHistory</u>	<u>+</u>	<u>+</u>	<u>MAY</u>	<u>+</u>	<u>+</u>	<u>n/a</u>
<u>getTaskInstanceData</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>n/a</u>
<u>getTaskOperations</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>
<u>hasSubtasks</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>n/a</u>
<u>instantiateSubTask</u>	<u>-</u>	<u>-</u>	<u>-</u>	<u>+</u>	<u>n/a</u>	<u>n/a</u>
<u>isSubtask</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>+</u>	<u>n/a</u>
<u>nominate</u>	<u>MAY</u>	<u>-</u>	<u>-</u>	<u>-</u>	<u>+</u>	<u>-</u>
<u>release</u>	<u>-</u>	<u>MAY</u>	<u>n/a</u>	<u>+</u>	<u>MAY</u>	<u>n/a</u>

Operation \ Role	Task Initiator	Task Stakeholders	Potential Owners	Actual Owner	Business Administrator	Notification Recipients
remove	-	n/a	n/a	n/a	+	+
resume	MAY	+	MAY	MAY	+	n/a
setFault	-	MAY	n/a	+	MAY	n/a
setGenericHumanRole	-	-	-	-	+	-
setOutput	-	MAY	n/a	+	MAY	n/a
setPriority	MAY	+	MAY	MAY	+	n/a
setTaskCompletionDeadlineExpression	MAY	+	-	-	+	n/a
setTaskCompletionDurationExpression	MAY	+	-	-	+	n/a
setTaskStartDeadlineExpression	MAY	+	-	-	+	n/a
setTaskStartDurationExpression	MAY	+	-	-	+	n/a
skip	+	+	MAY	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

Operation \ Role	Task Initiator	Task Stakeholders	Potential Owners	Actual Owner	Business Administrator	Notification Recipients
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
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*	+	+	+	+	+	+

2742

2743 7.2 XPath Extension Functions

2744 This section introduces XPath extension functions that are provided to be used within the definition of a
 2745 human task or notification. A WS-HumanTask Processor MUST support the XPath Functions listed below.
 2746 When defining properties using these XPath functions, note the initialization order in section 4.10.1.

2747 Definition of these XPath extension functions is provided in the table below. Input parameters that specify
 2748 task name, message part name or logicalPeopleGroup name MUST be literal strings. This restriction
 2749 does not apply to other parameters. Because XPath 1.0 functions do not support returning faults, an
 2750 empty node set is returned in the event of an error.

2751 XPath functions used for notifications in an escalation can access context from the enclosing task by
 2752 specifying that task's name.

2753
2754
2755

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 • task name (optional) Out • 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 • task name (optional) Out • 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 • task name (optional) Out • 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 • task name (optional) Out • 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 • task name (optional) Out • business administrators (<code>htt:organizationalEntity</code>)
getCountOfFinishedSubTasks getExcludedOwners	Returns the excluded owners. It MUST evaluate to an empty <code>htt:organizationalEntity</code>	In • task name (optional) Out

Operation Name	Description	Parameters
	<p>entity in case number of an error. finished sub tasks of a task</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)Number of the finished task sub-tasks. If the task doesn't have sub tasks then 0 is returned
getCountOfSubTasksgetTaskPriority	<p>Returns the priority number of the sub tasks of a task. 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:Priority)Number of the task sub-tasks. If the task doesn't have sub tasks then 0 is returned
getInputgetCountOfSubTasksInState	<p>Returns the part number of a task subtasks that are in the task's input message. specified state</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 state task name (optional) <p>Out</p> <ul style="list-style-type: none"> input message partNumber of the task sub tasks in the specified state. If the task doesn't have sub tasks then 0 is returned
getCountOfSubTasksWithOutcome	<p>Returns the number of a task sub tasks that match the given outcome</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"> outcome task name (optional) <p>Out</p> <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
getExcludedOwners	<p>Returns the excluded owners. It MUST evaluate to an empty htt:organizationalEntity in case of an error.</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"> excluded owners (htt:organizationalEntity)
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>

Operation Name	Description	Parameters
		<ul style="list-style-type: none"> <u>input message part</u>
<u>getLogicalPeopleGroup</u>	<p>Returns the value of a logical people group. In case of an error (e.g., when referencing a non existing logical people group) the <u>htt:organizationalEntity</u> MUST contain an empty user list.</p> <p>If the task name is not present the current task MUST be considered.</p>	<p><u>In</u></p> <ul style="list-style-type: none"> <u>name of the logical people group</u> <u>The optional parameters that follow MUST appear in pairs. Each pair is defined as:</u> <ul style="list-style-type: none"> <u>the qualified name of a logical people group parameter</u> <u>the value for the named logical people group parameter; it can be an XPath expression</u> <p><u>Out</u></p> <ul style="list-style-type: none"> <u>the value of the logical people group</u> (<u>htt:organizationalEntity</u>)
<u>getOutcome</u>	<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><u>In</u></p> <ul style="list-style-type: none"> <u>task name (optional)</u> <p><u>Out</u></p> <ul style="list-style-type: none"> <u>the task outcome</u> (<u>xsd:string</u>)
<u>getOutput</u>	<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><u>In</u></p> <ul style="list-style-type: none"> <u>part name</u> <u>task name (optional)</u> <p><u>Out</u></p> <ul style="list-style-type: none"> <u>output message part</u>
<u>getPotentialOwners</u>	<p>Returns the potential owners of the task. It MUST evaluate to an empty <u>htt:organizationalEntity</u> in case of an error.</p> <p>If the task name is not present the current task MUST be considered.</p>	<p><u>In</u></p> <ul style="list-style-type: none"> <u>task name (optional)</u> <p><u>Out</u></p> <ul style="list-style-type: none"> <u>potential owners</u> (<u>htt:organizationalEntity</u>)
<u>getSubtaskOutput</u>	<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</p>	<p><u>In</u></p> <ul style="list-style-type: none"> sub task name part name location path

Operation Name	Description	Parameters
	task MUST be considered	Out <ul style="list-style-type: none"> node-set of output message element(s)
getSubtaskOutputs	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	In <ul style="list-style-type: none"> part name location path Out <ul style="list-style-type: none"> node-set of output message elements from sub tasks
getOutput getTaskInitiator	Returns the part initiator of the task's output message task . It MUST evaluate to an empty htt:user 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"> part name task name (optional) Out <ul style="list-style-type: none"> output message partthe task initiator (user id as htt:user)
getTaskPriority getCountOfSubTasks	Returns the number priority of sub-tasks-of-a the task. It MUST evaluate to "5" in case the priority is not explicitly set. 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 priority (htt:tPriority)
getTaskStakeholders getCountOfFinishedSubTasks	Returns the number stakeholders of finished sub-tasks the task. It MUST evaluate to an empty htt:organizationalEntity in case of a task 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"> Number of the finished task sub-tasks. If the task doesn't have sub-tasks then 0 is returned task stakeholders (htt:organizationalEntity)
getCountOfSubTasksInSt	Returns the number of a task subtasks that are in	In

Operation Name	Description	Parameters
ate	<p>the specified state</p> <p>If the task name is not present the current task MUST be considered</p>	<ul style="list-style-type: none"> ● state ● task name (optional) <p>Out</p> <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	<p>Returns the number of a task sub tasks that match the given outcome</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"> ● outcome ● task name (optional) <p>Out</p> <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	<p>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.</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"> ● 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 XPath expression <p>Out</p> <ul style="list-style-type: none"> ● the value of the logical people group (<code>http: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</p>	<p>In</p> <ul style="list-style-type: none"> ● set1 (<code>http:organizationalEntity</code> <code>http:user</code>)

Operation Name	Description	Parameters
	users.	<ul style="list-style-type: none"> • set2 (htt:organizationalEntity htt:user) <p>Out</p> <ul style="list-style-type: none"> • result (htt:organizationalEntity)
intersect	Constructs an organizationalEntity containing every user that occurs in both set1 and set2 , eliminating duplicate users.	<p>In</p> <ul style="list-style-type: none"> • set1 (htt:organizationalEntity htt:user) • set2 (htt:organizationalEntity htt:user) <p>Out</p> <ul style="list-style-type: none"> • result (htt:organizationalEntity)

2758

2759

Generic set functions:

<u>Operation Name</u>	<u>Description</u>	<u>Parameters</u>
except	Constructs an organizationalEntity containing every user that occurs in set1 but not in set2 . Note: This function is required to allow enforcing the separation of duties ("4-eyes principle").	<p>In</p> <ul style="list-style-type: none"> • set1 (htt:organizationalEntity htt:user) • set2 (htt:organizationalEntity htt:user) <p>Out</p> <ul style="list-style-type: none"> • result (htt:organizationalEntity)
<u>intersect</u>	<u>Constructs an organizationalEntity containing every user that occurs in both set1 and set2, eliminating duplicate users.</u>	<p><u>In</u></p> <ul style="list-style-type: none"> • <u>set1</u> (<u>htt:organizationalEntity</u> <u> htt:user)</u> • <u>set2</u> (<u>htt:organizationalEntity</u> <u> htt:user)</u> <p><u>Out</u></p> <ul style="list-style-type: none"> • <u>result</u> (<u>htt:organizationalEntity</u> <u>)</u>

)
<u>union</u>	<u>Constructs an organizationalEntity containing every user that occurs in either set1 or set2, eliminating duplicate users.</u>	<u>In</u> <ul style="list-style-type: none"> <u>set1</u> (htt:organizationalEntity htt:user) <u>set2</u> (htt:organizationalEntity htt:user) <u>Out</u> <ul style="list-style-type: none"> <u>result</u> (htt:organizationalEntity)

2760

2761 In addition to the general-purpose functions listed above, the following aggregation functions MUST be
 2762 supported by a WS-HumanTask Processor. All aggregation functions take a node-set of strings,
 2763 booleans, or numbers as the first input parameter, and produce a result of the same type.

2764

2765 String-valued aggregation functions:

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	In <ul style="list-style-type: none"> node-set of string nodes percentage

Operation Name	Description	Parameters
	empty node-set)	

2766

2767

Boolean-valued aggregation functions:

<u>Operation Name</u>	<u>Description</u>	<u>Parameters</u>
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
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

2768

2769

Number-valued aggregation functions:

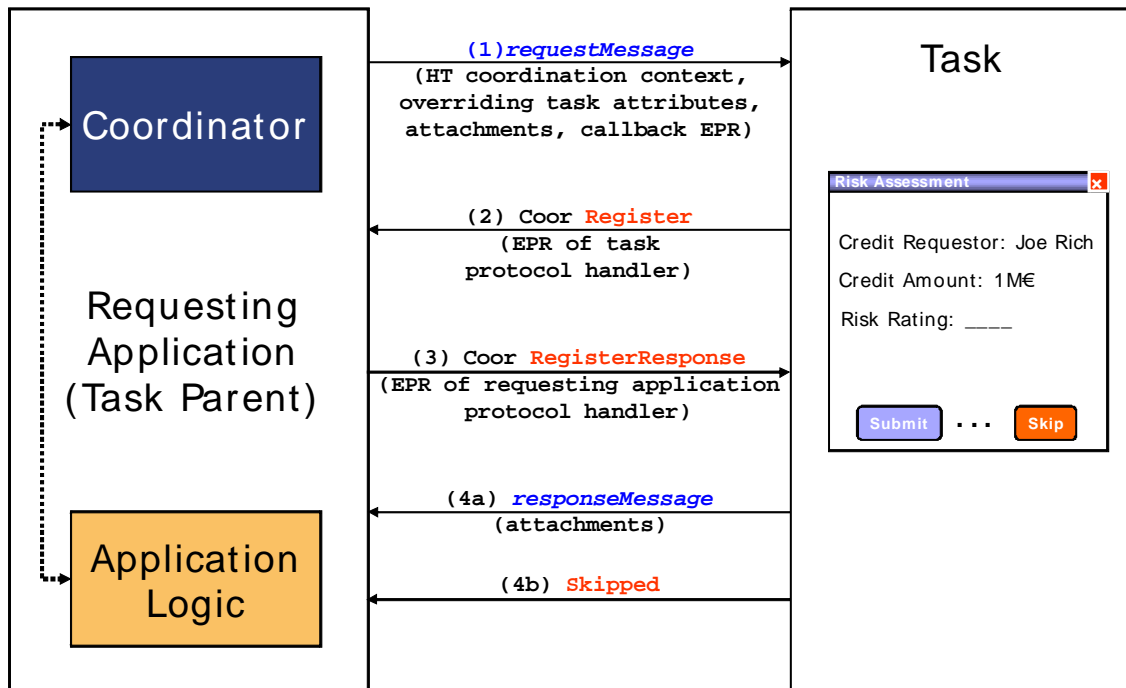
<u>Operation Name</u>	<u>Description</u>	<u>Parameters</u>
avg	Returns the average value of all number nodes - returns NaN for an empty node-set	In <ul style="list-style-type: none"> node-set of number nodes
max	Returns the maximum value of all number nodes - returns NaN for an empty node-set	In <ul 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 NaN for an empty node-set	In <ul style="list-style-type: none"> node-set of number nodes

2770

2771
2772
2773
2774
2775
2776
2777
2778
2779

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.



2780

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

2781 While we do not make any assumptions about the nature of the application in the following scenarios, in
2782 practice it would be hosted by an infrastructure that actually deals with the WS-HumanTask coordination
2783 protocol on the application's behalf.

2784 In case of human tasks the following message exchanges are possible.

2785 **Scenario 1:** At some point in time, the application invokes the human task through its service interface. In
2786 order to signal to the WS-HumanTask Processor that an instance of the human task can be created
2787 which is actually coordinated by the parent application, this request message contains certain control
2788 information. This control information consists of a coordination context of the WS-HumanTask
2789 coordination protocol, and optional human task attributes that are used to override aspects of the human
2790 task definition.

- 2791 • The coordination context (see [WS-C] for more details on Web services coordination framework
2792 used here) contains the element `CoordinationType` that MUST specify the WS-HumanTask
2793 coordination type [http://docs.oasis-open.org/ns/bpel4people/ws-](http://docs.oasis-open.org/ns/bpel4people/ws-humantask/protocol/200803)
2794 [humantask/protocol/200803](http://docs.oasis-open.org/ns/bpel4people/ws-humantask/protocol/200803). The inclusion of a coordination context within the request

2795 message indicates that the life cycle of the human tasks is managed via corresponding protocol
2796 messages from outside the WS-HumanTask Processor. The coordination context further contains
2797 in its `RegistrationService` element an endpoint reference that the WS-HumanTask
2798 Processor MUST use to register the task as a participant of that coordination type.
2799 Note: In a typical implementation, the parent application or its environment will create that
2800 coordination context by issuing an appropriate request against the WS-Coordination (WS-C)
2801 activation service, followed by registering the parent application as a `TaskParent` participant in
2802 that protocol.

- 2803 • The optional human task attributes allow overriding aspects of the definition of the human task
2804 from the calling application. The WS-HumanTask Parent MAY set values of the following
2805 attributes of the task definition:
 - 2806 ○ Priority of the task
 - 2807 ○ Actual people assignments for each of the generic human roles of the human task
 - 2808 ○ The skipable indicator which determines whether a task can actually be skipped at
2809 runtime.
 - 2810 ○ The amount of time by which the task activation is deferred.
 - 2811 ○ The expiration time for the human task after which the calling application is no longer
2812 interested in its result.

2813 After having created this request message, it is sent to the WS-HumanTask Processor (step (1) in Figure
2814 10). The WS-HumanTask Processor receiving that message MUST extract the coordination context and
2815 callback information, the human task attributes (if present) and the application payload. Before applying
2816 this application payload to the new human task, the WS-HumanTask Processor MUST register the human
2817 task to be created with the registration service passed as part of the coordination context (step (2) in
2818 Figure 10). The corresponding WS-C `Register` message MUST include the endpoint reference (EPR) of
2819 the protocol handler of the WS-HumanTask Processor that the WS-HumanTask Parent MUST use to
2820 send all protocol messages to WS-HumanTask Processor. This EPR is the value contained in the
2821 `ParticipantProtocolService` element of the `Register` message. Furthermore, the registration
2822 MUST be as a `HumanTask` participant by specifying the corresponding value in the
2823 `ProtocolIdentifier` element of the `Register` message. The WS-HumanTask Parent reacts to that
2824 message by sending back a `RegisterResponse` message. This message MUST contain in its
2825 `CoordinatorProtocolService` element the EPR of the protocol handler of the parent application,
2826 which MUST be used by the WS-HumanTask Processor for sending protocol messages to the parent
2827 application (step (3) in Figure 10).

2828 Now the instance of the human task is activated by the WS-HumanTask Processor, so the assigned
2829 person can perform the task (e.g. the risk assessment). Once the human task is successfully completed,
2830 a response message MUST be passed back to the parent application (step (4a) in Figure 10) by WS-
2831 HumanTask Processor.

2832 **Scenario 2:** If the human task is not completed with a result, but the assigned person determines that the
2833 task can be skipped (and hence reaches its *Obsolete* final state), then a “skipped” coordination protocol
2834 message MUST be sent from the WS-HumanTask Processor to its parent application (step (4b) in Figure
2835 10). No response message is passed back.

2836 **Scenario 3:** If the WS-HumanTask Parent needs to end prematurely before the invoked human task has
2837 been completed, it MUST send an `exit` coordination protocol message to the WS-HumanTask
2838 Processor causing the WS-HumanTask Processor to end its processing. A response message SHOULD
2839 NOT be passed back by WS-HumanTask Processor.

2840 In case of notifications to WS-HumanTask Processor, only some of the overriding attributes are
2841 propagated with the request message. Only priority and people assignments MAY be overridden for a
2842 notification, and the elements `isSkipable`, `expirationTime` and `attachments` MUST be ignored if present by
2843 WS-HumanTask Processor. Likewise, the WS-HumanTask coordination context, `attachments` and the
2844 callback EPR do not apply to notifications and MUST be ignored as well by WS-HumanTask Processor.
2845 Finally, a notification SHOULD NOT return WS-HumanTask coordination protocol messages. There
2846 SHOULD NOT be a message exchange beyond the initiating request message between the WS-
2847 HumanTask Processor and WS-HumanTask Parent.

2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882

8.1 Human Task Coordination Protocol Messages

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

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

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

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

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

2883 8.2 Protocol Messages

2884 All WS-HumanTask protocol messages have the following type:

```
2885 <xsd:complexType name="tProtocolMsgType">
2886   <xsd:sequence>
2887     <xsd:any namespace="##other" processContents="lax"
2888       minOccurs="0" maxOccurs="unbounded" />
2889   </xsd:sequence>
2890   <xsd:anyAttribute namespace="##other" processContents="lax" />
2891 </xsd:complexType>
```

2892 This message type is extensible and any implementation MAY use this extension mechanism to define
2893 proprietary attributes and content which are out of the scope of this specification.

2894 8.2.1 Protocol Messages Received by a Task Parent

2895 The following is the definition of the `htcp:skipped` message.

```
2896 <xsd:element name="skipped" type="htcp:tProtocolMsgType" />
2897 <wsdl:message name="skipped">
2898   <wsdl:part name="parameters" element="htcp:skipped" />
2899 </wsdl:message>
```

2900 The `htcp:skipped` message is used to inform the task parent (i.e. the requesting application) that the
2901 invoked task has been skipped. The task does not return any result.

2902 The following is the definition of the `htcp:fault` message.

```
2903 <xsd:element name="fault" type="htcp:tProtocolMsgType" />
2904 <wsdl:message name="fault">
2905   <wsdl:part name="parameters" element="htcp:fault" />
2906 </wsdl:message>
```

2907 The `htcp:fault` message is used to inform the task parent that the task has ended abnormally. The
2908 task does not return any result.

2909 8.2.2 Protocol Messages Received by a Task

2910 Upon receipt of the following `htcp:exit` message the task parent informs the task that it is no longer
2911 interested in its results.

```
2912 <xsd:element name="exit" type="htcp:tProtocolMsgType" />
2913 <wsdl:message name="exit">
2914   <wsdl:part name="parameters" element="htcp:exit" />
2915 </wsdl:message>
```

2916 8.3 WSDL of the Protocol Endpoints

2917 Protocol messages are received by protocol participants via operations of dedicated ports called protocol
2918 endpoints. In this section we specify the WSDL port types of the protocol endpoints needed to run the
2919 WS-HumanTask coordination protocol.

2920 8.3.1 Protocol Endpoint of the Task Parent

2921 An application that wants to create a task and wants to become a task parent MUST provide an endpoint
2922 implementing the following port type. This endpoint is the protocol endpoint of the task parent receiving
2923 protocol messages of the WS-HumanTask coordination protocol from a task. The operation used by the
2924 task to send a certain protocol message to the task parent is named by the message name of the protocol
2925 message concatenated by the string `Operation`. For example, the `skipped` message MUST be passed
2926 to the task parent by using the operation named `skippedOperation`.

```
2927 <wsdl:portType name="clientParticipantPortType">
```

```

2928 <wsdl:operation name="skippedOperation">
2929   <wsdl:input message="htcp:skipped" />
2930 </wsdl:operation>
2931 <wsdl:operation name="faultOperation">
2932   <wsdl:input message="htcp:fault" />
2933 </wsdl:operation>
2934 </wsdl:portType>

```

2935 8.3.2 Protocol Endpoint of the Task

2936 For a WS-HumanTask Definition a task MUST provide an endpoint implementing the following port type.
 2937 This endpoint is the protocol endpoint of the task receiving protocol messages of the WS-HumanTask
 2938 coordination protocol from a task parent. The operation used by the task parent to send a certain protocol
 2939 message to a task is named by the message name of the protocol message concatenated by the string
 2940 Operation. For example, the exit protocol message MUST be passed to the task by using the
 2941 operation named exitOperation.

```

2942 <wsdl:portType name="humanTaskParticipantPortType">
2943   <wsdl:operation name="exitOperation">
2944     <wsdl:input message="htcp:exit" />
2945   </wsdl:operation>
2946 </wsdl:portType>

```

2947 8.4 Providing Human Task Context

2948 The task context information is exchanged between the requesting application and a task or a notification.
 2949 In case of tasks, this information is passed as header fields of the request and response messages of the
 2950 task's operation. In case of notifications, this information is passed as header fields of the request
 2951 message of the notification's operation.

2952 8.4.1 SOAP Binding of Human Task Context

2953 In general, a SOAP binding specifies for message header fields how they are bound to SOAP headers. In
 2954 case of WS-HumanTask, the humanTaskRequestContext and humanTaskResponseContext
 2955 elements are simply mapped to SOAP header as a whole. The following listings show the SOAP binding
 2956 of the human task request context and human task response context in an infoset representation.

```

2957 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
2958           xmlns:htc="http://docs.oasis-open.org/ns/bpel4people/ws-
2959 humantask/context/200803">
2960   <S:Header>
2961     <htc:humanTaskRequestContext>
2962       <htc:priority>...</htc:priority>?
2963       <htc:attachments>...</htc:attachments>?
2964       <htc:peopleAssignments>...</htc:peopleAssignments>?
2965       <htc:isSkipable>...</htc:isSkipable>?
2966       <htc:activationDeferralTime>...</htc:activationDeferralTime>?
2967       <htc:expirationTime>...</htc:expirationTime>?
2968       ... extension elements ...
2969     </htc:humanTaskRequestContext>
2970   </S:Header>
2971   <S:Body>
2972     ...
2973   </S:Body>
2974 </S:Envelope>

```

```

2975
2976 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"

```



```

2977     xmlns:htc="http://docs.oasis-open.org/ns/bpel4people/ws-
2978 humantask/context/200803">
2979     <S:Header>
2980       <htc:humanTaskResponseContext>
2981         <htc:priority>...</htc:priority>?
2982         <htc:attachments>...</htc:attachments>?
2983         <htc:actualOwner>...</htc:actualOwner>?
2984         <htc:actualPeopleAssignments>...</htc:actualPeopleAssignments>?
2985         <htc:outcome>...</htc:outcome>?
2986         ... extension elements ...
2987       </htc:humanTaskResponseContext>
2988     </S:Header>
2989     <S:Body>
2990       ...
2991     </S:Body>
2992 </S:Envelope>

```

2993 The following listing is an example of a SOAP message containing a human task request context.

```

2994 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
2995     xmlns:htc="http://docs.oasis-open.org/ns/bpel4people/ws-
2996 humantask/context/200803">
2997   <S:Header>
2998     <htc:humanTaskRequestContext>
2999       <htc:priority>0</htc:priority>
3000       <htc:peopleAssignments>
3001         <htc:potentialOwners>
3002           <htt:organizationalEntity>
3003             <htt:user>Alan</htt:user>
3004             <htt:user>Dieter</htt:user>
3005             <htt:user>Frank</htt:user>
3006             <htt:user>Gerhard</htt:user>
3007             <htt:user>Ivana</htt:user>
3008             <htt:user>Karsten</htt:user>
3009             <htt:user>Matthias</htt:user>
3010             <htt:user>Patrick</htt:user>
3011           </htt:organizationalEntity>
3012         </htc:potentialOwners>
3013       </htc:peopleAssignments>
3014     </htc:humanTaskRequestContext>
3015   </S:Header>
3016   <S:Body>...</S:Body>
3017 </S:Envelope>

```

3018 8.4.2 Overriding Task Definition People Assignments

3019 The task context information exchanged contains a `potentialOwners` element, which can be used at
3020 task creation time to override the set of task assignments that we defined in the original task definition.
3021 Compliant implementations MUST allow overriding of simple tasks and routing patterns that are a single-
3022 level deep, i.e. routing patterns that don't have nested routing patterns. If the task context
3023 `potentialOwners` contains a list of `htt:user` and `htt:group`, and the task definition contains a
3024 routing pattern element `htt:parallel` or `htt:sequence` that has as its only children `htt:user` and
3025 `htt:group` elements, the WS-HumanTask Processor MUST replace the list in the task definition with the
3026 list in the task context. If the task definition contains only a list of `htt:user` and `htt:group`, then the
3027 WS-HumanTask Processor MUST replace the list of users from the task definition with the list of users in
3028 the task context.

3029 8.5 Human Task Policy Assertion

3030 In order to support discovery of Web services that support the human task contract that are available for
3031 coordination by another service, a *human task policy* assertion is defined by WS-HumanTask. This policy
3032 assertion can be associated with the business operation used by the invoking component (recall that the
3033 human task is restricted to have exactly one business operation). In doing so, the provider of a human
3034 task can signal whether or not the corresponding task can communicate with an invoking component via
3035 the WS-HumanTask coordination protocol.

3036 The following describes the policy assertion used to specify that an operation can be used to instantiate a
3037 human task with the proper protocol in place:

```
3038 <http:HumanTaskAssertion wsp:Optional="true"? ...>  
3039 ...  
3040 </http:HumanTaskAssertion>
```

3041 /http:HumanTaskAssertion

3042 This policy assertion specifies that the WS-HumanTask Parent, in this case the sender, **MUST**
3043 include context information for a human task coordination type passed with the message. The
3044 receiving human task **MUST** be instantiated with the WS-Human Task protocol in place by the
3045 WS-HumanTask Processor.

3046 /http:HumanTaskAssertion/@wsp:Optional="true"

3047 As defined in WS-Policy [WS-Policy], this is the compact notation for two policy alternatives, one
3048 with and one without the assertion. Presence of both policy alternatives indicates that the
3049 behavior indicated by the assertion is optional, such that a WS-HumanTask coordination context
3050 **MAY** be passed with an input message. If the context is passed the receiving human task **MUST**
3051 be instantiated with the WS-HumanTask protocol in place. The absence of the assertion is
3052 interpreted to mean that a WS-HumanTask coordination context **SHOULD NOT** be passed with
3053 an input message.

3054 The human task policy assertion indicates behavior for a single operation, thus the assertion has an
3055 Operation Policy Subject. WS-PolicyAttachment [WS-PolAtt] defines two policy attachment points with
3056 Operation Policy Subject, namely wsdl:portType/wsdl:operation and wsdl:binding/wsdl:operation.

3057 The <http:HumanTaskAssertion> policy assertion can also be used for notifications. In that case it
3058 means that the WS-HumanTask Parent, in this case the sender, **MAY** pass the human task context
3059 information with the message. Other headers, including headers with the coordination context are
3060 ignored.

3061 9 Task Parent Interactions with Lean Tasks

3062 9.1 Operations for Task Parent Applications

3063 A number of operations are involved in the life cycle of a lean task definition. These comprise:

- 3064 • Registering a lean task definition, such that it is available for later use
- 3065 • Unregistering a lean task definition, such that it is no longer available for later use
- 3066 • Listing lean task definitions, to determine what is available for use
- 3067 • Creating a lean task from a lean task definition

3068 An operation takes a well-defined set of parameters as its input. Passing an illegal parameter or an illegal
3069 number of parameters MUST result in the `htlt:illegalArgumentFault` being returned. Invoking an
3070 operation that is not allowed in the current state of the lean task definition MUST result in an
3071 `htlt:illegalStateFault`.

3072 By default, the identity of the person on behalf of which the operation is invoked is passed to the WS-
3073 HumanTask Processor. When the person is not authorized to perform the operation the
3074 `htlt:illegalAccessFault` MUST be returned.

3075 This specification does not stipulate the authentication, addressing, and binding scheme employed when
3076 calling an operation. This can be achieved using different mechanisms (e.g. WS-Security, WS-
3077 Addressing).

3078 9.2 Lean Task Interactions

3079 To enable lightweight task definition and creation by a WS-HumanTask Parent, a conformant WS-
3080 HumanTask Processor MUST provide the following operations:

- 3081 • `registerLeanTaskDefinition` API for registration
- 3082 • `unregisterLeanTaskDefinition` API for retraction
- 3083 • `listLeanTaskDefinitions` API for enumeration
- 3084 • `createLeanTask` and `createLeanTaskAsync` APIs for creation

3085 and invoke the following callback operation in response to `createLeanTaskAsync`:

- 3086 • `createLeanTaskAsyncCallback`

3087 9.2.1 Register a Lean Task Definition

```
3088 <xsd:element name="registerLeanTaskDefinition">
3089   <xsd:complexType>
3090     <xsd:sequence>
3091       <xsd:element name="taskDefinition" type="htd:tLeanTask" />
3092     </xsd:sequence>
3093   </xsd:complexType>
3094 </xsd:element>
3095 <xsd:element name="registerLeanTaskDefinitionResponse">
3096   <xsd:complexType>
3097     <xsd:sequence>
3098       <xsd:element name="taskName" type="xsd:NCName" />
3099     </xsd:sequence>
3100   </xsd:complexType>
3101 </xsd:element>
```

3102 The `htlt:registerLeanTaskDefinition` operation is used to create a new Lean Task definition that
3103 is available for future listing and consumption by the `htlt:listLeanTaskDefinitions` and
3104 `htlt:createLeanTask` / `htlt:createLeanTaskAsync` operations. If an existing Lean Task exists at
3105 the same name as the `htd:tLeanTask/@Name`, the WSHumanTask Processor SHOULD return an
3106 `htlt:illegalStateFault`.

3107 9.2.2 Unregister a Lean Task Definition

```
3108 <xsd:element name="unregisterLeanTaskDefinition">  
3109   <xsd:complexType>  
3110     <xsd:sequence>  
3111       <xsd:element name="taskName" type="xsd:NCName" />  
3112     </xsd:sequence>  
3113   </xsd:complexType>  
3114 </xsd:element>  
3115 <xsd:element name="unregisterLeanTaskDefinitionResponse">  
3116   <xsd:complexType>  
3117     <xsd:sequence>  
3118       <xsd:element name="taskName" type="xsd:NCName" />  
3119     </xsd:sequence>  
3120   </xsd:complexType>  
3121 </xsd:element>
```

3122 The `htlt:unregisterLeanTaskDefinition` operation is used to remove a Lean Task available for
3123 future listing and consumption by the `htlt:listLeanTaskDefinitions` and
3124 `htlt:createLeanTask` / `htlt:createLeanTaskAsync` operations. The WS-HumanTask Processor
3125 SHOULD also move any instances of lean tasks of this task definition to "Error" state. If the Lean Task
3126 does not already exist as a registered element, the WS-HumanTask Processor MUST return an
3127 `htlt:illegalArgumentFault`.

3128 9.2.3 List Lean Task Definitions

```
3129 <xsd:element name="listLeanTaskDefinitions">  
3130   <xsd:complexType>  
3131     <xsd:sequence>  
3132       <xsd:annotation>  
3133         <xsd:documentation>Empty message</xsd:documentation>  
3134       </xsd:annotation>  
3135     </xsd:sequence>  
3136   </xsd:complexType>  
3137 </xsd:element>  
3138 <xsd:element name="listLeanTaskDefinitionsResponse">  
3139   <xsd:complexType>  
3140     <xsd:sequence>  
3141       <xsd:element name="leanTaskDefinitions">  
3142         <xsd:complexType>  
3143           <xsd:sequence>  
3144             <xsd:element name="leanTaskDefinition" type="htd:tLeanTask"  
3145 minOccurs="0" maxOccurs="unbounded" />  
3146           </xsd:sequence>  
3147         </xsd:complexType>  
3148       </xsd:element>  
3149     </xsd:sequence>  
3150   </xsd:complexType>  
3151 </xsd:element>
```

3152 The `htlt:listLeanTaskDefinitions` operation is used to query the list of `htd:tLeanTask`
3153 elements that are registered Lean Tasks, as registered by the `htlt:registerLeanTaskDefinition`
3154 operation, and not subsequently unregistered by `htlt:unregisterLeanTaskDefinition`.

3155 9.2.4 Create a Lean Task

```
3156 <xsd:element name="CreateLeanTask">  
3157   <xsd:complexType>  
3158     <xsd:sequence>  
3159       <xsd:element name="inputMessage">  
3160         <xsd:complexType>  
3161           <xsd:sequence>  
3162             <xsd:any processContents="lax" namespace="##any" />  
3163           </xsd:sequence>  
3164         </xsd:complexType>  
3165       </xsd:element>  
3166       <xsd:element name="taskDefinition" type="htd:tLeanTask" minOccurs="0"/>  
3167       <xsd:element name="taskName" type="xsd:NCName" minOccurs="0" />  
3168     </xsd:sequence>  
3169   </xsd:complexType>  
3170 </xsd:element>  
3171 <xsd:element name="CreateLeanTaskResponse">  
3172   <xsd:complexType>  
3173     <xsd:sequence>  
3174       <xsd:element name="outputMessage">  
3175         <xsd:complexType>  
3176           <xsd:sequence>  
3177             <xsd:any processContents="lax" namespace="##any" />  
3178           </xsd:sequence>  
3179         </xsd:complexType>  
3180       </xsd:element>  
3181     </xsd:sequence>  
3182   </xsd:complexType>  
3183 </xsd:element>  
3184
```

3185 The `htlt:createLeanTask` operation is called by a WS-HumanTask Parent to create a task based on
3186 a Lean Task definition. This task definition either can be passed in directly to the operation or can
3187 reference a Lean Task definition previously sent via `htlt:registerLeanTaskDefinition`. These
3188 tasks follow the standard pattern of the Human Task Coordination protocol and is the operation on the
3189 portType used to create a task in that standard pattern, using the `humanTaskRequestContext` and
3190 `humanTaskResponseContext` as described in section 8.4.

3191 If both `taskName` and `taskDefinition` are set, the WS-HumanTask Processor MUST return an
3192 `htlt:illegalArgumentFault`. If `taskName` is set and a lean task has been registered by that name,
3193 the WS-HumanTask Process MUST use the registered lean task definition to create the task. If `taskName`
3194 is not set and a lean task has not been registered by that name, the WS-HumanTask Processor MUST
3195 return an `htlt:illegalArgumentFault`. If `taskDefinition` is set, the WS-HumanTask Processor MUST
3196 use the `taskDefinition` element as the type of the task to create. The WS-HumanTask Processor MUST
3197 use the `inputMessage` as the input message of the task and return the output message of the task in
3198 the `outputMessage` element.

3199 The `htlt:createLeanTask` operation is long-running because its execution includes the user
3200 interaction with the task owner. As a result, it is not meaningful to bind the request-response operation to
3201 a protocol that blocks any resources until the response is returned.

3202 Alternatively, instead of invoking the long-running request-response operation defined above, an
3203 interaction style using an asynchronous callback operation can be used. In this case, the WS-HumanTask
3204 Parent invokes the following `htlt:createLeanTaskAsync` operation and, as described in section 10,

3205 passes a WS-Addressing endpoint reference (EPR) in order to provide a callback address for delivering
3206 the lean task's output.

3207 Technically, `htlt:createLeanTaskAsync` is also a request-response operation in order to enable
3208 returning faults, but it returns immediately to the caller if the lean task is created successfully, without
3209 waiting for the lean task to complete.

```
3210 <xsd:element name="createLeanTaskAsync">  
3211   <xsd:complexType>  
3212     <xsd:sequence>  
3213       <xsd:element name="inputMessage">  
3214         <xsd:complexType>  
3215           <xsd:sequence>  
3216             <xsd:any processContents="lax" namespace="##any" />  
3217           </xsd:sequence>  
3218         </xsd:complexType>  
3219       </xsd:element>  
3220       <xsd:element name="taskDefinition" type="htd:tLeanTask" minOccurs="0"/>  
3221       <xsd:element name="taskName" type="xsd:NCName" minOccurs="0" />  
3222     </xsd:sequence>  
3223   </xsd:complexType>  
3224 </xsd:element>  
3225 <xsd:element name="createLeanTaskAsyncResponse">  
3226   <xsd:complexType>  
3227     <xsd:sequence/>  
3228   </xsd:complexType>  
3229 </xsd:element>
```

3230 Upon completion of the lean task, the WS-HumanTask Processor invokes the callback operation
3231 `htlt:createLeanTaskAsyncCallback` at the callback address specified in the EPR passed by the
3232 WS-HumanTask Parent.

```
3233 <xsd:element name="createLeanTaskAsyncCallback">  
3234   <xsd:complexType>  
3235     <xsd:sequence>  
3236       <xsd:element name="outputMessage">  
3237         <xsd:complexType>  
3238           <xsd:sequence>  
3239             <xsd:any processContents="lax" namespace="##any" />  
3240           </xsd:sequence>  
3241         </xsd:complexType>  
3242       </xsd:element>  
3243     </xsd:sequence>  
3244   </xsd:complexType>  
3245 </xsd:element>
```

3246 9.2.5 Endpoints for Lean Task Operations

3247 A WS-HumanTask Processor MUST provide an endpoint implementing the following port type. This
3248 endpoint is used to register, unregister, and list lean task definitions, and create a lean task given a
3249 particular definition and input message.

```
3250 <wsdl:portType name="leanTaskOperations">  
3251   <wsdl:operation name="registerLeanTaskDefinition">  
3252     <wsdl:input message="registerLeanTaskDefinition" />  
3253     <wsdl:output message="registerLeanTaskDefinitionResponse" />  
3254     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />  
3255     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />  
3256   </wsdl:operation>  
3257 </wsdl:portType>  
3258
```

```

3259 <wsdl:operation name="unregisterLeanTaskDefinition">
3260   <wsdl:input message="unregisterLeanTaskDefinition" />
3261   <wsdl:output message="unregisterLeanTaskDefinitionResponse" />
3262   <wsdl:fault name="illegalArgumentFault" message="illegalArgumentFault" />
3263   <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
3264 </wsdl:operation>
3265
3266 <wsdl:operation name="listLeanTaskDefinitions">
3267   <wsdl:input message="listLeanTaskDefinitions" />
3268   <wsdl:output message="listLeanTaskDefinitionsResponse" />
3269   <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
3270 </wsdl:operation>
3271
3272 <wsdl:operation name="createLeanTask">
3273   <wsdl:input message="createLeanTask" />
3274   <wsdl:output message="createLeanTaskResponse" />
3275   <wsdl:fault name="illegalArgumentFault" message="illegalArgumentFault" />
3276   <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
3277 </wsdl:operation>
3278
3279 <wsdl:operation name="createLeanTaskAsync">
3280   <wsdl:input message="createLeanTaskAsync" />
3281   <wsdl:output message="createLeanTaskAsyncResponse" />
3282   <wsdl:fault name="illegalArgumentFault" message="illegalArgumentFault" />
3283   <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
3284 </wsdl:operation>
3285
3286 </wsdl:portType>

```

3287 A WS-HumanTask Parent invoking the `htlt:createLeanTaskAsync` operation MUST provide an
3288 endpoint implementing the following callback port type.

```

3289 <wsdl:portType name="leanTaskCallbackOperations">
3290
3291   <wsdl:operation name="createLeanTaskAsyncCallback">
3292     <wsdl:input message="createLeanTaskAsyncCallback" />
3293   </wsdl:operation>
3294
3295 </wsdl:portType>

```


3296

10 Providing Callback Information for Human Tasks

3297 WS-HumanTask extends the information model of a WS-Addressing endpoint reference (EPR) defined in
3298 [WS-Addr-Core] (see [WS-Addr-SOAP] and [WS-Addr-WSDL] for more details). This extension is needed
3299 to support passing information to human tasks about ports and operations of a caller receiving responses
3300 from such human tasks.

3301 Passing this callback information from a WS-HumanTask Parent (i.e. a requesting application) to the WS-
3302 HumanTask Processor MAY override static deployment information that may have been set.

10.1 EPR Information Model Extension

3304 Besides the properties of an endpoint reference (EPR) defined by [WS-Addr-Core] WS-HumanTask
3305 defines the following abstract properties:

3306 [response action] : xsd:anyURI (0..1)

3307 This property contains the value of the [action] message addressing property to be sent within the
3308 response message.

3309 [response operation] : xsd:NCName (0..1)

3310 This property contains the name of a WSDL operation.

3311 Each of these properties is a child element of the [metadata] property of an endpoint reference. An
3312 endpoint reference passed by a caller to a WS-HumanTask Processor MUST contain the [metadata]
3313 property. Furthermore, this [metadata] property MUST contain either a [response action] property or a
3314 [response operation] property.

3315 If present, the value of the [response action] property MUST be used by the WS-HumanTask Processor
3316 hosting the responding human task to specify the value of the [action] message addressing property of
3317 the response message sent back to the caller. Furthermore, the [destination] property of this response
3318 message MUST be copied from the [address] property of the EPR contained in the original request
3319 message by the WS-HumanTask Processor.

3320 If present, the value of the [response operation] property MUST be the name of an operation of the port
3321 type implemented by the endpoint denoted by the [address] property of the EPR. The corresponding port
3322 type MUST be included as a WSDL 1.1 definition nested within the [metadata] property of the EPR (see
3323 [WS-Addr-WSDL]). The WS-HumanTask Processor hosting the responding human task MUST use the
3324 value of the [response operation] property as operation of the specified port type at the specified endpoint
3325 to send the response message. Furthermore, the [metadata] property MUST contain WSDL 1.1 binding
3326 information corresponding to the port type implemented by the endpoint denoted by the [address]
3327 property of the EPR.

3328 The EPR sent from the caller to the WS-HumanTask Processor MUST identify the instance of the caller.
3329 This MUST be done by the caller in one of the two ways: First, the value of the [address] property can
3330 contain a URL with appropriate parameters uniquely identifying the caller instance. Second, appropriate
3331 [reference parameters] properties are specified within the EPR. The values of these [reference
3332 parameters] uniquely identify the caller within the scope of the URI passed within the [address] property.

10.2 XML Infoset Representation

3334 The following describes the infoset representation of the EPR extensions introduced by WS-HumanTask:

```
3335 <wsa:EndpointReference>  
3336   <wsa:Address>xsd:anyURI</wsa:Address>  
3337   <wsa:ReferenceParameters>xsd:any*</wsa:ReferenceParameters>?  
3338   <wsa:Metadata>  
3339     <htcp:responseAction>xsd:anyURI</htcp:responseAction>?  
3340     <htcp:responseOperation>xsd:NCName</htcp:responseOperation>?  
3341   </wsa:Metadata>
```


3342 `</wsa:EndpointReference>`
3343 `/wsa:EndpointReference/wsa:Metadata`
3344 This element of the EPR MUST be sent by WS-HumanTask Parent, the caller, to the WS-
3345 HumanTask Processor . It MUST either contain WSDL 1.1 metadata specifying the information to
3346 access the endpoint (i.e. its port type, bindings or ports) according to [WS-Addr-WSDL] as well as
3347 a `<htcp:responseOperation>` element, or it MUST contain a `<htcp:responseAction>`
3348 element.

3349 `/wsa:EndpointReference/wsa:Metadata/htcp:responseAction`
3350 This element (of type `xsd:anyURI`) specifies the value of the [action] message addressing
3351 property to be used by the receiving WS-HumanTask Processor when sending the response
3352 message from the WS-HumanTask Processor back to the caller. If this element is specified the
3353 `<htcp:responseOperation>` element MUST NOT be specified by the caller.

3354 `/wsa:EndpointReference/wsa:Metadata/htcp:responseOperation`
3355 This element (of type `xsd:NCName`) specifies the name of the operation that MUST be used by
3356 the receiving WS-HumanTask Processor to send the response message from the WS-
3357 HumanTask Processor back to the caller.. If this element is specified the
3358 `<htcp:responseAction>` element MUST NOT be specified by the WS-HumanTask Parent.

3359 Effectively, WS-HumanTask defines two ways to pass callback information from the caller to the human
3360 task. First, the EPR contains just the value of the [action] message addressing property that MUST be
3361 used by the WS-HumanTask Processor within the response message (i.e. the
3362 `<htcp:responseAction>` element). Second, the EPR contains the WSDL 1.1 metadata for the port
3363 receiving the response operation. In this case, for the callback information the WS-HumanTask Parent
3364 MUST specify which operation of that port is to be used (i.e. the `<htcp:responseOperation>`
3365 element). In both cases, the response is typically sent to the address specified in the `<wsa:Address>`
3366 element of the EPR contained in the original request message; note, that [WS-Addr-WSDL] does not
3367 exclude redirection to other addresses than the one specified, but the corresponding mechanisms are out
3368 of the scope of the specification.

3369 The following example of an endpoint reference shows the usage of the `<htcp:responseAction>`
3370 element. The `<wsa:Metadata>` elements contain the `<htcp:responseAction>` element that
3371 specifies the value of the [action] message addressing property to be used by the WS-HumanTask
3372 Processor when sending the response message back to the caller. This value is
3373 `http://example.com/LoanApproval/approvalResponse`. The value of the [destination] message
3374 addressing property to be used is given in the `<wsa:Address>` element, namely
3375 `http://example.com/LoanApproval/loan?ID=42`. Note that this URL includes the HTTP search
3376 part with the parameter `ID=42` which uniquely identifies the instance of the caller.

```
3377 <wsa:EndpointReference
3378   xmlns:wsa="http://www.w3.org/2005/08/addressing">
3379
3380   <wsa:Address>http://example.com/LoanApproval/loan?ID=42</wsa:Address>
3381
3382   <wsa:Metadata>
3383     <htcp:responseAction>
3384       http://example.com/LoanApproval/approvalResponse
3385     </htcp:responseAction>
3386   </wsa:Metadata>
3387
3388 </wsa:EndpointReference>
```

3389 The following example of an endpoint reference shows the usage of the `<htcp:responseOperation>`
3390 element and corresponding WSDL 1.1 metadata. The port type of the caller that receives the response
3391 message from the WS-HumanTask Processor is defined using the `<wsdl:portType>` element. In our
3392 example it is the `LoanApprovalPT` port type. The definition of the port type is nested in a corresponding
3393 WSLD 1.1 `<wsdl:definitions>` element in the `<wsa:Metadata>` element. This

3394 <wsdl:definitions> element also contains a binding for this port type as well as a corresponding
3395 port definition nested in a <wsdl:service> element. The <http:responseOperation> element
3396 specifies that the approvalResponse operation of the LoanApprovalPT port type is used to send the
3397 response to the caller. The address of the actual port to be used which implements the
3398 LoanApprovalPT port type and thus the approvalResponse operation is given in the
3399 <wsa:Address> element, namely the URL http://example.com/LoanApproval/loan. The
3400 unique identifier of the instance of the caller is specified in the <xmp:MyInstanceID> element nested in
3401 the <wsa:ReferenceParameters> element.

```
3402 <wsa:EndpointReference
3403   xmlns:wsa="http://www.w3.org/2005/08/addressing">
3404
3405   <wsa:Address>http://example.com/LoanApproval/loan</wsa:Address>
3406
3407   <wsa:ReferenceParameters>
3408     <xmp:MyInstanceID>42</xmp:MyInstanceID>
3409   </wsa:ReferenceParameters>
3410
3411   <wsa:Metadata>
3412
3413     <wsdl:definitions ...>
3414
3415       <wsdl:portType name="LoanApprovalPT">
3416         <wsdl:operation name="approvalResponse">...</wsdl:operation>
3417         ...
3418       </wsdl:portType>
3419
3420       <wsdl:binding name="LoanApprovalSoap" type="LoanApprovalPT">
3421         ...
3422       </wsdl:binding>
3423
3424       <wsdl:service name="LoanApprovalService">
3425         <wsdl:port name="LA" binding="LoanApprovalSoap">
3426           <soap:address
3427             location="http://example.com/LoanApproval/loan" />
3428         </wsdl:port>
3429         ...
3430       </wsdl:service>
3431
3432     </wsdl:definitions>
3433
3434     <http:responseOperation>approvalResponse</http:responseOperation>
3435
3436   </wsa:Metadata>
3437
3438 </wsa:EndpointReference>
```

3439 **10.3 Message Addressing Properties**

3440 Message addressing properties provide references for the endpoints involved in an interaction at the
3441 message level. For this case, WS-HumanTask Processor uses the message addressing properties
3442 defined in [WS-Addr-Core] for the request message as well as for the response message.

3443 The request message sent by the caller (i.e. the requesting application) to the human task uses the
3444 message addressing properties as described in [WS-Addr-Core]. WS-HumanTask refines the use of the
3445 following message addressing properties:

- 3446 • The [reply endpoint] message addressing property MUST contain the EPR to be used by the WS-
3447 HumanTask Processor to send its response to.

3448 Note that the [fault endpoint] property MUST NOT be used by WS-HumanTask Processor. This is
3449 because via one-way operation no application level faults are returned to the caller.

3450 The response message sent by the WS-HumanTask Processor to the caller uses the message
3451 addressing properties as defined in [WS-Addr-Core] and refines the use of the following properties:

- 3452 • The value of the [action] message addressing property is set as follows:
 - 3453 • If the original request message contains the `<htcp:responseAction>` element in the
3454 `<wsa:Metadata>` element of the EPR of the [reply endpoint] message addressing property,
3455 the value of the former element MUST be copied into the [action] property of the response
3456 message by WS-HumanTask Processor.
 - 3457 • If the original request message contains the `<htcp:responseOperation>` element (and,
3458 thus, WSDL 1.1 metadata) in the `<wsa:Metadata>` element of the EPR of the [reply
3459 endpoint] message addressing property, the value of the [action] message addressing
3460 property of the response message is determined as follows:
 - 3461 • Assume that the WSDL 1.1 metadata specifies within the binding chosen a value for the
3462 `soapaction` attribute on the `soap:operation` element of the response operation.
3463 Then, this value MUST be used as value of the [action] property by WS-HumanTask
3464 Processor.
 - 3465 • If no such `soapaction` attribute is provided, the value of the [action] property MUST be
3466 derived as specified in [WS-Addr-WSDL] by WS-HumanTask Processor.
- 3467 • Reference parameters are mapped as specified in [WS-Addr-SOAP].

3468 10.4 SOAP Binding

3469 A SOAP binding specifies how abstract message addressing properties are bound to SOAP headers. In
3470 this case, WS-HumanTask Processor MUST use the mappings as specified by [WS-Addr-SOAP].

3471 The following is an example of a request message sent from the caller to the WS-HumanTask Processor
3472 containing the `<htcp:responseAction>` element in the incoming EPR. The EPR is mapped to SOAP
3473 header fields as follows: The endpoint reference to be used by the human task for submitting its response
3474 message to is contained in the `<wsa:ReplyTo>` element. The address of the endpoint is contained in the
3475 `<wsa:Address>` element. The identifier of the instance of the caller to be encoded as reference
3476 parameters in the response message is nested in the `<wsa:ReferenceParameters>` element. The
3477 value of the `<wsa:Action>` element to be set by the human task in its response to the caller is in the
3478 `<htcp:responseAction>` element nested in the `<wsa:Metadata>` element of the EPR.

```
3479 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"  
3480   xmlns:wsa="http://www.w3.org/2005/08/addressing"  
3481   xmlns:htcp="http://docs.oasis-open.org/ns/bpel4people/ws-  
3482 humantask/protocol/200803">  
3483  
3484   <S:Header>  
3485     <wsa:ReplyTo>  
3486       <wsa:Address>http://example.com/LoanApproval/loan</wsa:Address>  
3487       <wsa:ReferenceParameters>  
3488         <xmp:MyInstanceID>42</xmp:MyInstanceID>  
3489       </wsa:ReferenceParameters>  
3490       <wsa:Metadata>  
3491         <htcp:responseAction>  
3492           http://example.com/LoanApproval/approvalResponse  
3493         </htcp:responseAction>  
3494       </wsa:Metadata>  
3495     </wsa:ReplyTo>  
3496   </S:Header>  
3497  
3498   <S:Body>...</S:Body>
```

3499 </S:Envelope>

3500 The following is an example of a response message corresponding to the request message discussed
3501 above. This response is sent from the WS-HumanTask Processor back to the caller. The <wsa:To>
3502 element contains a copy of the <wsa:Address> element of the original request message. The
3503 <wsa:Action> element is copied from the <htcp:responseAction> element of the original request
3504 message. The reference parameters are copied as standalone elements (the <xmp:MyInstanceID>
3505 element below) out of the <wsa:ReferenceParameters> element of the request message.

```
3506 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"  
3507   xmlns:wsa="http://www.w3.org/2005/08/addressing">  
3508   <S:Header>  
3509     <wsa:To>  
3510       <wsa:Address>http://example.com/LoanApproval/loan</wsa:Address>  
3511     </wsa:To>  
3512     <wsa:Action>  
3513       http://example.com/LoanApproval/approvalResponse  
3514     </wsa:Action>  
3515     <xmp:MyInstanceID wsa:IsReferenceParameter='true'>  
3516       42  
3517     </xmp:MyInstanceID>  
3518   </S:Header>  
3519   <S:Body>...</S:Body>  
3520 </S:Envelope>
```

3521 The following is an example of a request message sent from the caller to the WS-HumanTask Processor
3522 containing the <htcp:responseOperation> element and corresponding WSDL metadata in the
3523 incoming EPR. The EPR is mapped to SOAP header fields as follows: The endpoint reference to be used
3524 by the WS-HumanTask Processor for submitting its response message to is contained in the
3525 <wsa:ReplyTo> element. The address of the endpoint is contained in the <wsa:Address> element.
3526 The identifier of the instance of the caller to be encoded as reference parameters in the response
3527 message is nested in the <wsa:ReferenceParameters> element. The WSDL metadata of the
3528 endpoint is contained in the <wsdl:definitions> element. The name of the operation of the endpoint
3529 to be used to send the response message to is contained in the <htcp:responseOperation>
3530 element. Both elements are nested in the <wsa:Metadata> element of the EPR. These elements
3531 provide the basis to determine the value of the action header field to be set by the WS-HumanTask
3532 Processor in its response to the caller.

```
3533 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"  
3534   xmlns:wsa="http://www.w3.org/2005/08/addressing"  
3535   xmlns:htcp="http://docs.oasis-open.org/ns/bpel4people/ws-  
3536   humantask/protocol/200803">  
3537   <S:Header>  
3538     <wsa:ReplyTo>  
3539       <wsa:Address>http://example.com/LoanApproval/loan</wsa:Address>  
3540     </wsa:ReplyTo>  
3541     <wsa:ReferenceParameters>  
3542       <xmp:MyInstanceID>42</xmp:MyInstanceID>  
3543     </wsa:ReferenceParameters>  
3544     <wsa:Metadata>  
3545       <wsdl:definitions  
3546         targetNamespace="http://example.com/loanApproval"  
3547         xmlns:wsdl="..." xmlns:soap="...">  
3548         <wsdl:portType name="LoanApprovalPT">  
3549           <wsdl:operation name="approvalResponse">  
3550             <wsdl:input name="approvalInput" ... />  
3551           </wsdl:operation>  
3552         </wsdl:portType>  
3553       </wsdl:definitions>  
3554     </wsa:Metadata>  
3555   </S:Header>  
3556   <S:Body>...</S:Body>  
3557 </S:Envelope>
```

```

3555     </wsdl:operation>
3556     ...
3557 </wsdl:portType>
3558
3559     <wsdl:binding name="LoanApprovalSoap"
3560         type="LoanApprovalPT">
3561         ...
3562     </wsdl:binding>
3563
3564     <wsdl:service name="LoanApprovalService">
3565         <wsdl:port name="LA" binding="LoanApprovalSoap">
3566             <soap:address
3567                 location="http://example.com/LoanApproval/loan" />
3568         </wsdl:port>
3569         ...
3570     </wsdl:service>
3571 </wsdl:definitions>
3572
3573     <htcp:responseOperation>
3574         approvalResponse
3575     </htcp:responseOperation>
3576
3577     </wsa:Metadata>
3578 </wsa:ReplyTo>
3579
3580 </S:Header>
3581 <S:Body>...</S:Body>
3582 </S:Envelope>

```

3583 The following is an example of a response message corresponding to the request message before; this
3584 response is sent from the WS-HumanTask Processor back to the caller. The `<wsa:To>` element contains
3585 a copy of the `<wsa:Address>` field of the original request message. The reference parameters are
3586 copied as standalone element (the `<xmp:MyInstanceID>` element below) out of the
3587 `<htcp:ReferenceParameters>` element of the request message. The value of the `<wsa:Action>`
3588 element is composed according to [WS-Addr-WSDL] from the target namespace, port type name, name
3589 of the response operation to be used, and name of the input message of this operation given in the code
3590 snippet above.

```

3591 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
3592     xmlns:wsa="http://www.w3.org/2005/08/addressing"
3593     xmlns:htd="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803">
3594 <S:Header>
3595     <wsa:To>http://example.com/LoanApproval/loan</wsa:To>
3596     <wsa:Action>
3597         http://example.com/loanApproval/...
3598         ...LoanApprovalPT/approvalResponse/ApprovalInput
3599     </wsa:Action>
3600     <xmp:MyInstanceID wsa:IsReferenceParameter='true'>
3601         42
3602     </xmp:MyInstanceID>
3603 </S:Header>
3604 <S:Body>...</S:Body>
3605 </S:Envelope>

```

3606

11 Security Considerations

3607 WS-HumanTask does not mandate the use of any specific mechanism or technology for client
3608 authentication. However, a client **MUST** provide a principal or the principal **MUST** be obtainable by the
3609 WS-HumanTask Processor.

3610 When using task APIs via SOAP bindings, compliance with the WS-I Basic Security Profile 1.0 is
3611 **RECOMMENDED**.

3612 **12 Conformance**

3613

3614 The XML schema pointed to by the RDDL document at the namespace URI, defined by this specification,
3615 are considered to be authoritative and take precedence over the XML schema defined in the appendix of
3616 this document.

3617

3618 There are four conformance targets defined as part of this specification: a WS-HumanTask Definition, a
3619 WS-HumanTask Processor, a WS-HumanTask Parent and a WS-HumanTask Client (see section 2.3). In
3620 order to claim conformance with WS-HumanTask 1.1, the conformance targetes MUST comply with all
3621 normative statements in this specification, notably all MUST statements have to be implemented.

3622

A. Portability and Interoperability Considerations

3623

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

3624

- Portability - The ability to take human tasks and notifications created in one vendor's environment and use them in another vendor's environment.

3625

3626

- 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.

3627

3628

3629

3630

Portability requires support of WS-HumanTask artifacts.

3631

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

3632

3633

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

3634

3635

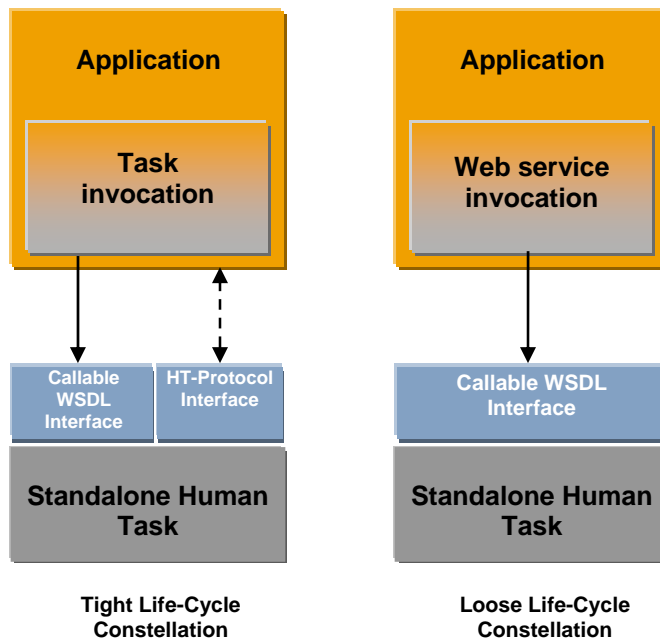
3636

Tight Life-Cycle Constellation: Applications are human task aware and control the life cycle of tasks.

3637

Interoperability between applications and WS-HumanTask Processors is achieved using the WS-HumanTask coordination protocol.

3638



3639

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

3640

3641

3642

B. WS-HumanTask Language Schema

```
3643 <?xml version="1.0" encoding="UTF-8"?>
3644 <!--
3645   Copyright (c) OASIS Open 2009. All Rights Reserved.
3646 -->
3647 <xsd:schema
3648   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3649   xmlns="http://docs.oasis-open.org/ns/bpel4people/ws-humanTask/200803"
3650   targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
3651   humanTask/200803"
3652   elementFormDefault="qualified" blockDefault="#all">
3653
3654   <xsd:annotation>
3655     <xsd:documentation>
3656       XML Schema for WS-HumanTask 1.1 - WS-HumanTask Task Definition Language
3657     </xsd:documentation>
3658   </xsd:annotation>
3659
3660   <!-- other namespaces -->
3661   <xsd:import namespace="http://www.w3.org/XML/1998/namespace"
3662     schemaLocation="http://www.w3.org/2001/xml.xsd" />
3663
3664   <!-- base types for extensible elements -->
3665   <xsd:complexType name="tExtensibleElements">
3666     <xsd:sequence>
3667       <xsd:element name="documentation" type="tDocumentation" minOccurs="0"
3668   maxOccurs="unbounded" />
3669       <xsd:any namespace="##other" processContents="lax" minOccurs="0"
3670   maxOccurs="unbounded" />
3671     </xsd:sequence>
3672     <xsd:anyAttribute namespace="##other" processContents="lax" />
3673   </xsd:complexType>
3674
3675   <xsd:complexType name="tDocumentation" mixed="true">
3676     <xsd:sequence>
3677       <xsd:any namespace="##other" processContents="lax" minOccurs="0"
3678   maxOccurs="unbounded" />
3679     </xsd:sequence>
3680     <xsd:attribute ref="xml:lang" />
3681   </xsd:complexType>
3682
3683   <xsd:complexType name="tExtensibleMixedContentElements"
3684     mixed="true">
3685     <xsd:sequence>
3686       <xsd:element name="documentation" type="tDocumentation" minOccurs="0"
3687   maxOccurs="unbounded" />
3688       <xsd:any namespace="##other" processContents="lax" minOccurs="0"
3689   maxOccurs="unbounded" />
3690     </xsd:sequence>
3691     <xsd:anyAttribute namespace="##other" processContents="lax" />
3692   </xsd:complexType>
3693
3694   <!-- human interactions definition -->
3695   <xsd:element name="humanInteractions" type="tHumanInteractions" />
3696   <xsd:complexType name="tHumanInteractions">
```

```

3697     <xsd:complexContent>
3698         <xsd:extension base="tExtensibleElements">
3699             <xsd:sequence>
3700                 <xsd:element name="extensions" type="tExtensions" minOccurs="0" />
3701                 <xsd:element name="import" type="tImport" minOccurs="0"
3702 maxOccurs="unbounded" />
3703                 <xsd:element name="logicalPeopleGroups" type="tLogicalPeopleGroups"
3704 minOccurs="0" />
3705                 <xsd:element name="tasks" type="tTasks" minOccurs="0" />
3706                 <xsd:element name="notifications" type="tNotifications"
3707 minOccurs="0" />
3708             </xsd:sequence>
3709             <xsd:attribute name="targetNamespace" type="xsd:anyURI"
3710 use="required" />
3711             <xsd:attribute name="queryLanguage" type="xsd:anyURI" />
3712             <xsd:attribute name="expressionLanguage" type="xsd:anyURI" />
3713         </xsd:extension>
3714     </xsd:complexContent>
3715 </xsd:complexType>
3716
3717 <xsd:complexType name="tExtensions">
3718     <xsd:complexContent>
3719         <xsd:extension base="tExtensibleElements">
3720             <xsd:sequence>
3721                 <xsd:element name="extension" type="tExtension"
3722 maxOccurs="unbounded" />
3723             </xsd:sequence>
3724         </xsd:extension>
3725     </xsd:complexContent>
3726 </xsd:complexType>
3727
3728 <xsd:complexType name="tExtension">
3729     <xsd:complexContent>
3730         <xsd:extension base="tExtensibleElements">
3731             <xsd:attribute name="namespace" type="xsd:anyURI" use="required" />
3732             <xsd:attribute name="mustUnderstand" type="tBoolean" use="required"
3733 />
3734         </xsd:extension>
3735     </xsd:complexContent>
3736 </xsd:complexType>
3737
3738 <xsd:element name="import" type="tImport" />
3739 <xsd:complexType name="tImport">
3740     <xsd:complexContent>
3741         <xsd:extension base="tExtensibleElements">
3742             <xsd:attribute name="namespace" type="xsd:anyURI" use="optional" />
3743             <xsd:attribute name="location" type="xsd:anyURI" use="optional" />
3744             <xsd:attribute name="importType" type="xsd:anyURI" use="required" />
3745         </xsd:extension>
3746     </xsd:complexContent>
3747 </xsd:complexType>
3748
3749 <xsd:element name="logicalPeopleGroups" type="tLogicalPeopleGroups" />
3750 <xsd:complexType name="tLogicalPeopleGroups">
3751     <xsd:complexContent>
3752         <xsd:extension base="tExtensibleElements">
3753             <xsd:sequence>

```

```

3754         <xsd:element name="logicalPeopleGroup" type="tLogicalPeopleGroup"
3755 maxOccurs="unbounded" />
3756     </xsd:sequence>
3757 </xsd:extension>
3758 </xsd:complexContent>
3759 </xsd:complexType>
3760
3761 <xsd:complexType name="tLogicalPeopleGroup">
3762 <xsd:complexContent>
3763 <xsd:extension base="tExtensibleElements">
3764 <xsd:sequence>
3765 <xsd:element name="parameter" type="tParameter" minOccurs="0"
3766 maxOccurs="unbounded" />
3767 </xsd:sequence>
3768 <xsd:attribute name="name" type="xsd:NCName" use="required" />
3769 <xsd:attribute name="reference" type="xsd:NCName" use="optional" />
3770 </xsd:extension>
3771 </xsd:complexContent>
3772 </xsd:complexType>
3773
3774 <!-- generic human roles used in tasks and notifications -->
3775 <xsd:element name="genericHumanRole" type="tGenericHumanRoleAssignmentBase"
3776 abstract="true" block="" />
3777
3778 <xsd:element name="potentialOwners" type="tPotentialOwnerAssignment"
3779 substitutionGroup="genericHumanRole" />
3780 <xsd:element name="excludedOwners" type="tGenericHumanRoleAssignment"
3781 substitutionGroup="genericHumanRole" />
3782 <xsd:element name="taskInitiator" type="tGenericHumanRoleAssignment"
3783 substitutionGroup="genericHumanRole" />
3784 <xsd:element name="taskStakeholders" type="tGenericHumanRoleAssignment"
3785 substitutionGroup="genericHumanRole" />
3786 <xsd:element name="businessAdministrators"
3787 type="tGenericHumanRoleAssignment" substitutionGroup="genericHumanRole" />
3788 <xsd:element name="recipients" type="tGenericHumanRoleAssignment"
3789 substitutionGroup="genericHumanRole" />
3790
3791 <xsd:complexType name="tGenericHumanRoleAssignmentBase" block="">
3792 <xsd:complexContent>
3793 <xsd:extension base="tExtensibleElements" />
3794 </xsd:complexContent>
3795 </xsd:complexType>
3796
3797 <xsd:complexType name="tGenericHumanRoleAssignment">
3798 <xsd:complexContent>
3799 <xsd:extension base="tGenericHumanRoleAssignmentBase">
3800 <xsd:sequence>
3801 <xsd:element name="from" type="tFrom" />
3802 </xsd:sequence>
3803 </xsd:extension>
3804 </xsd:complexContent>
3805 </xsd:complexType>
3806
3807 <xsd:complexType name="tPotentialOwnerAssignment">
3808 <xsd:complexContent>
3809 <xsd:extension base="tGenericHumanRoleAssignmentBase">
3810 <xsd:choice>
3811 <xsd:element name="from" type="tFrom" />

```

```

3812     <xsd:element name="parallel" type="tParallel" />
3813     <xsd:element name="sequence" type="tSequence" />
3814   </xsd:choice>
3815 </xsd:extension>
3816 </xsd:complexContent>
3817 </xsd:complexType>
3818
3819 <!-- routing patterns -->
3820 <xsd:complexType name="tParallel">
3821   <xsd:complexContent>
3822     <xsd:extension base="tExtensibleElements">
3823       <xsd:sequence>
3824         <xsd:element name="completionBehavior" type="tCompletionBehavior"
3825 minOccurs="0" />
3826         <xsd:element name="from" type="tFrom" minOccurs="0"
3827 maxOccurs="unbounded" />
3828         <xsd:choice minOccurs="0" maxOccurs="unbounded">
3829           <xsd:element name="parallel" type="tParallel" />
3830           <xsd:element name="sequence" type="tSequence" />
3831         </xsd:choice>
3832       </xsd:sequence>
3833       <xsd:attribute name="type" type="tRoutingPatternType" />
3834     </xsd:extension>
3835   </xsd:complexContent>
3836 </xsd:complexType>
3837
3838 <xsd:complexType name="tSequence">
3839   <xsd:complexContent>
3840     <xsd:extension base="tExtensibleElements">
3841       <xsd:sequence>
3842         <xsd:element name="completionBehavior" type="tCompletionBehavior"
3843 />
3844         <xsd:element name="from" type="tFrom" minOccurs="0"
3845 maxOccurs="unbounded" />
3846         <xsd:choice minOccurs="0" maxOccurs="unbounded">
3847           <xsd:element name="parallel" type="tParallel" />
3848           <xsd:element name="sequence" type="tSequence" />
3849         </xsd:choice>
3850       </xsd:sequence>
3851       <xsd:attribute name="type" type="tRoutingPatternType" />
3852     </xsd:extension>
3853   </xsd:complexContent>
3854 </xsd:complexType>
3855
3856 <xsd:simpleType name="tRoutingPatternType">
3857   <xsd:restriction base="xsd:string">
3858     <xsd:enumeration value="all" />
3859     <xsd:enumeration value="single" />
3860   </xsd:restriction>
3861 </xsd:simpleType>
3862
3863 <!-- completion behavior -->
3864 <xsd:complexType name="tCompletionBehavior">
3865   <xsd:complexContent>
3866     <xsd:extension base="tExtensibleElements">
3867       <xsd:sequence>
3868         <xsd:element name="completion" type="tCompletion" minOccurs="0"
3869 maxOccurs="unbounded" />

```

```

3870         <xsd:element name="defaultCompletion" type="tDefaultCompletion"
3871 minOccurs="0" />
3872     </xsd:sequence>
3873     <xsd:attribute name="completionAction" type="tPattern" use="optional"
3874 default="automatic" />
3875 </xsd:extension>
3876 </xsd:complexContent>
3877 </xsd:complexType>
3878
3879 <xsd:complexType name="tCompletion">
3880 <xsd:complexContent>
3881 <xsd:extension base="tExtensibleElements">
3882 <xsd:sequence>
3883 <xsd:element name="condition" type="tBoolean-expr" />
3884 <xsd:element name="result" type="tResult" minOccurs="0" />
3885 </xsd:sequence>
3886 </xsd:extension>
3887 </xsd:complexContent>
3888 </xsd:complexType>
3889
3890 <xsd:complexType name="tDefaultCompletion">
3891 <xsd:complexContent>
3892 <xsd:extension base="tExtensibleElements">
3893 <xsd:sequence>
3894 <xsd:element name="result" type="tResult" />
3895 </xsd:sequence>
3896 </xsd:extension>
3897 </xsd:complexContent>
3898 </xsd:complexType>
3899
3900 <!-- result construction -->
3901 <xsd:complexType name="tResult">
3902 <xsd:complexContent>
3903 <xsd:extension base="tExtensibleElements">
3904 <xsd:choice maxOccurs="unbounded">
3905 <xsd:element name="aggregate" type="tAggregate" />
3906 <xsd:element name="copy" type="tCopy" />
3907 </xsd:choice>
3908 </xsd:extension>
3909 </xsd:complexContent>
3910 </xsd:complexType>
3911
3912 <xsd:complexType name="tAggregate">
3913 <xsd:complexContent>
3914 <xsd:extension base="tExtensibleElements">
3915 <xsd:attribute name="part" type="xsd:NCName" use="optional" />
3916 <xsd:attribute name="location" type="xsd:string" use="optional" />
3917 <xsd:attribute name="condition" type="xsd:string" />
3918 <xsd:attribute name="function" type="xsd:string" use="required" />
3919 </xsd:extension>
3920 </xsd:complexContent>
3921 </xsd:complexType>
3922
3923 <xsd:complexType name="tCopy">
3924 <xsd:complexContent>
3925 <xsd:extension base="tExtensibleElements">
3926 <xsd:sequence>
3927 <xsd:element name="from" type="tExpression" />

```

```

3928     <xsd:element name="to" type="tQuery" />
3929   </xsd:sequence>
3930 </xsd:extension>
3931 </xsd:complexContent>
3932 </xsd:complexType>
3933
3934 <!-- human tasks -->
3935 <xsd:element name="tasks" type="tTasks" />
3936 <xsd:complexType name="tTasks">
3937   <xsd:complexContent>
3938     <xsd:extension base="tExtensibleElements">
3939       <xsd:sequence>
3940         <xsd:element name="task" type="tTask" maxOccurs="unbounded" />
3941       </xsd:sequence>
3942     </xsd:extension>
3943   </xsd:complexContent>
3944 </xsd:complexType>
3945
3946 <xsd:complexType name="tTaskBase" abstract="true">
3947   <xsd:complexContent>
3948     <xsd:extension base="tExtensibleElements">
3949       <xsd:sequence>
3950         <xsd:element name="interface" type="tTaskInterface" minOccurs="0"
3951 />
3952         <xsd:element name="messageSchema" type="tMessageSchema"
3953 minOccurs="0" />
3954         <xsd:element name="priority" type="tPriority-expr" minOccurs="0" />
3955         <xsd:element name="peopleAssignments" type="tPeopleAssignments"
3956 minOccurs="0" />
3957         <xsd:element name="completionBehavior" type="tCompletionBehavior"
3958 minOccurs="0" />
3959         <xsd:element name="delegation" type="tDelegation" minOccurs="0" />
3960         <xsd:element name="presentationElements"
3961 type="tPresentationElements" minOccurs="0" />
3962         <xsd:element name="outcome" type="tQuery" minOccurs="0" />
3963         <xsd:element name="searchBy" type="tExpression" minOccurs="0" />
3964         <xsd:element name="renderings" type="tRenderings" minOccurs="0" />
3965         <xsd:element name="deadlines" type="tDeadlines" minOccurs="0" />
3966         <xsd:element name="composition" type="tComposition" minOccurs="0"
3967 />
3968       </xsd:sequence>
3969       <xsd:attribute name="name" type="xsd:NCName" use="required" />
3970       <xsd:attribute name="actualOwnerRequired" type="tBoolean"
3971 use="optional" default="yes" />
3972     </xsd:extension>
3973   </xsd:complexContent>
3974 </xsd:complexType>
3975
3976 <xsd:element name="task" type="tTask" />
3977 <xsd:complexType name="tTask">
3978   <xsd:complexContent>
3979     <xsd:restriction base="tTaskBase">
3980       <xsd:sequence>
3981         <xsd:element name="documentation" type="tDocumentation"
3982 minOccurs="0" maxOccurs="unbounded" />
3983         <xsd:any namespace="##other" processContents="lax" minOccurs="0"
3984 maxOccurs="unbounded" />
3985         <xsd:element name="interface" type="tTaskInterface" />

```



```

3986     <xsd:element name="messageSchema" type="tMessageSchema"
3987 minOccurs="0" maxOccurs="0" />
3988     <xsd:element name="priority" type="tPriority-expr" minOccurs="0" />
3989     <xsd:element name="peopleAssignments" type="tPeopleAssignments"
3990 minOccurs="0" />
3991     <xsd:element name="completionBehavior" type="tCompletionBehavior"
3992 minOccurs="0" />
3993     <xsd:element name="delegation" type="tDelegation" minOccurs="0" />
3994     <xsd:element name="presentationElements"
3995 type="tPresentationElements" minOccurs="0" />
3996     <xsd:element name="outcome" type="tQuery" minOccurs="0" />
3997     <xsd:element name="searchBy" type="tExpression" minOccurs="0" />
3998     <xsd:element name="renderings" type="tRenderings" minOccurs="0" />
3999     <xsd:element name="deadlines" type="tDeadlines" minOccurs="0" />
4000     <xsd:element name="composition" type="tComposition" minOccurs="0"
4001 />
4002     </xsd:sequence>
4003     <xsd:attribute name="name" type="xsd:NCName" use="required" />
4004     <xsd:attribute name="actualOwnerRequired" type="tBoolean"
4005 use="optional" default="yes" />
4006     <xsd:anyAttribute namespace="##other" processContents="lax" />
4007     </xsd:restriction>
4008   </xsd:complexContent>
4009 </xsd:complexType>
4010
4011 <xsd:complexType name="tTaskInterface">
4012   <xsd:complexContent>
4013     <xsd:extension base="tExtensibleElements">
4014       <xsd:attribute name="portType" type="xsd:QName" use="required" />
4015       <xsd:attribute name="operation" type="xsd:NCName" use="required" />
4016       <xsd:attribute name="responsePortType" type="xsd:QName"
4017 use="optional" />
4018       <xsd:attribute name="responseOperation" type="xsd:NCName"
4019 use="optional" />
4020     </xsd:extension>
4021   </xsd:complexContent>
4022 </xsd:complexType>
4023
4024 <!-- presentation elements -->
4025 <xsd:complexType name="tPresentationElements">
4026   <xsd:complexContent>
4027     <xsd:extension base="tExtensibleElements">
4028       <xsd:sequence>
4029         <xsd:element name="name" type="tText" minOccurs="0"
4030 maxOccurs="unbounded" />
4031         <xsd:element name="presentationParameters"
4032 type="tPresentationParameters" minOccurs="0" />
4033         <xsd:element name="subject" type="tText" minOccurs="0"
4034 maxOccurs="unbounded" />
4035         <xsd:element name="description" type="tDescription" minOccurs="0"
4036 maxOccurs="unbounded" />
4037       </xsd:sequence>
4038     </xsd:extension>
4039   </xsd:complexContent>
4040 </xsd:complexType>
4041
4042 <xsd:complexType name="tPresentationParameters">
4043   <xsd:complexContent>

```

```

4044     <xsd:extension base="tExtensibleElements">
4045         <xsd:sequence>
4046             <xsd:element name="presentationParameter"
4047 type="tPresentationParameter" maxOccurs="unbounded" />
4048         </xsd:sequence>
4049         <xsd:attribute name="expressionLanguage" type="xsd:anyURI" />
4050     </xsd:extension>
4051 </xsd:complexContent>
4052 </xsd:complexType>
4053
4054 <xsd:complexType name="tPresentationParameter">
4055     <xsd:complexContent>
4056         <xsd:extension base="tParameter" />
4057     </xsd:complexContent>
4058 </xsd:complexType>
4059
4060 <!-- elements for rendering tasks -->
4061 <xsd:complexType name="tRenderings">
4062     <xsd:complexContent>
4063         <xsd:extension base="tExtensibleElements">
4064             <xsd:sequence>
4065                 <xsd:element name="rendering" type="tRendering"
4066 maxOccurs="unbounded" />
4067             </xsd:sequence>
4068         </xsd:extension>
4069     </xsd:complexContent>
4070 </xsd:complexType>
4071
4072 <xsd:complexType name="tRendering">
4073     <xsd:complexContent>
4074         <xsd:extension base="tExtensibleElements">
4075             <xsd:attribute name="type" type="xsd:QName" use="required" />
4076         </xsd:extension>
4077     </xsd:complexContent>
4078 </xsd:complexType>
4079
4080 <!-- elements for people assignment -->
4081 <xsd:element name="peopleAssignments" type="tPeopleAssignments" />
4082 <xsd:complexType name="tPeopleAssignments">
4083     <xsd:complexContent>
4084         <xsd:extension base="tExtensibleElements">
4085             <xsd:sequence>
4086                 <xsd:element ref="genericHumanRole" minOccurs="0"
4087 maxOccurs="unbounded" />
4088             </xsd:sequence>
4089         </xsd:extension>
4090     </xsd:complexContent>
4091 </xsd:complexType>
4092
4093 <!-- elements for handling timeouts and escalation -->
4094 <xsd:complexType name="tDeadlines">
4095     <xsd:complexContent>
4096         <xsd:extension base="tExtensibleElements">
4097             <xsd:sequence>
4098                 <xsd:element name="startDeadline" type="tDeadline" minOccurs="0"
4099 maxOccurs="unbounded" />
4100                 <xsd:element name="completionDeadline" type="tDeadline"
4101 minOccurs="0" maxOccurs="unbounded" />

```



```

4102     </xsd:sequence>
4103     </xsd:extension>
4104     </xsd:complexContent>
4105 </xsd:complexType>
4106
4107 <xsd:complexType name="tDeadline">
4108     <xsd:complexContent>
4109         <xsd:extension base="tExtensibleElements">
4110             <xsd:sequence>
4111                 <xsd:choice>
4112                     <xsd:element name="for" type="tDuration-expr" />
4113                     <xsd:element name="until" type="tDeadline-expr" />
4114                 </xsd:choice>
4115                 <xsd:element name="escalation" type="tEscalation" minOccurs="0"
4116 minOccurs="unbounded" />
4117             </xsd:sequence>
4118             <xsd:attribute name="name" type="xsd:NCName" use="required" />
4119         </xsd:extension>
4120     </xsd:complexContent>
4121 </xsd:complexType>
4122
4123 <xsd:complexType name="tEscalation">
4124     <xsd:complexContent>
4125         <xsd:extension base="tExtensibleElements">
4126             <xsd:sequence>
4127                 <xsd:element name="condition" type="tBoolean-expr" minOccurs="0" />
4128                 <xsd:element name="toParts" type="tToParts" minOccurs="0" />
4129             <xsd:choice>
4130                 <xsd:element name="notification" type="tNotification" />
4131                 <xsd:element name="localNotification" type="tLocalNotification"
4132 />
4133                 <xsd:element name="reassignment" type="tReassignment" />
4134             </xsd:choice>
4135         </xsd:sequence>
4136         <xsd:attribute name="name" type="xsd:NCName" use="required" />
4137     </xsd:extension>
4138 </xsd:complexContent>
4139 </xsd:complexType>
4140
4141 <xsd:complexType name="tLocalNotification">
4142     <xsd:complexContent>
4143         <xsd:extension base="tExtensibleElements">
4144             <xsd:choice>
4145                 <xsd:sequence>
4146                     <xsd:element name="priority" type="tPriority-expr" minOccurs="0"
4147 />
4148                     <xsd:element name="peopleAssignments" type="tPeopleAssignments"
4149 minOccurs="0" />
4150                 </xsd:sequence>
4151             </xsd:choice>
4152             <xsd:attribute name="reference" type="xsd:QName" use="required" />
4153         </xsd:extension>
4154     </xsd:complexContent>
4155 </xsd:complexType>
4156
4157 <xsd:complexType name="tReassignment">
4158     <xsd:complexContent>
4159         <xsd:extension base="tExtensibleElements">

```

```

4160     <xsd:sequence>
4161         <xsd:element ref="potentialOwners" />
4162     </xsd:sequence>
4163 </xsd:extension>
4164 </xsd:complexContent>
4165 </xsd:complexType>
4166
4167 <xsd:complexType name="tToParts">
4168     <xsd:complexContent>
4169         <xsd:extension base="tExtensibleElements">
4170             <xsd:sequence>
4171                 <xsd:element name="toPart" type="tToPart" maxOccurs="unbounded" />
4172             </xsd:sequence>
4173         </xsd:extension>
4174     </xsd:complexContent>
4175 </xsd:complexType>
4176
4177 <xsd:complexType name="tToPart" mixed="true">
4178     <xsd:complexContent>
4179         <xsd:extension base="tExtensibleMixedContentElements">
4180             <xsd:attribute name="name" type="xsd:NCName" use="required" />
4181             <xsd:attribute name="expressionLanguage" type="xsd:anyURI" />
4182         </xsd:extension>
4183     </xsd:complexContent>
4184 </xsd:complexType>
4185
4186 <!-- task delegation -->
4187 <xsd:complexType name="tDelegation">
4188     <xsd:complexContent>
4189         <xsd:extension base="tExtensibleElements">
4190             <xsd:sequence>
4191                 <xsd:element name="from" type="tFrom" minOccurs="0" />
4192             </xsd:sequence>
4193             <xsd:attribute name="potentialDelegates" type="tPotentialDelegates"
4194 use="required" />
4195         </xsd:extension>
4196     </xsd:complexContent>
4197 </xsd:complexType>
4198
4199 <xsd:simpleType name="tPotentialDelegates">
4200     <xsd:restriction base="xsd:string">
4201         <xsd:enumeration value="anybody" />
4202         <xsd:enumeration value="nobody" />
4203         <xsd:enumeration value="potentialOwners" />
4204         <xsd:enumeration value="other" />
4205     </xsd:restriction>
4206 </xsd:simpleType>
4207
4208 <!-- composite tasks -->
4209 <xsd:complexType name="tComposition">
4210     <xsd:complexContent>
4211         <xsd:extension base="tExtensibleElements">
4212             <xsd:sequence>
4213                 <xsd:element name="subtask" type="tSubtask" maxOccurs="unbounded"
4214 />
4215             </xsd:sequence>
4216             <xsd:attribute name="type" type="tCompositionType" use="optional"
4217 default="sequential" />

```

```

4218         <xsd:attribute name="instantiationPattern" type="tPattern"
4219 use="optional" default="manual" />
4220     </xsd:extension>
4221 </xsd:complexContent>
4222 </xsd:complexType>
4223
4224 <xsd:simpleType name="tCompositionType">
4225     <xsd:restriction base="xsd:string">
4226         <xsd:enumeration value="sequential" />
4227         <xsd:enumeration value="parallel" />
4228     </xsd:restriction>
4229 </xsd:simpleType>
4230
4231 <xsd:simpleType name="tPattern">
4232     <xsd:restriction base="xsd:string">
4233         <xsd:enumeration value="manual" />
4234         <xsd:enumeration value="automatic" />
4235     </xsd:restriction>
4236 </xsd:simpleType>
4237
4238 <xsd:complexType name="tSubtask">
4239     <xsd:complexContent>
4240         <xsd:extension base="tExtensibleElements">
4241             <xsd:choice>
4242                 <xsd:element name="task" type="tTask"/>
4243                 <xsd:element name="localTask" type="tLocalTask" />
4244             </xsd:choice>
4245             <xsd:attribute name="name" type="xsd:NCName" use="required" />
4246         </xsd:extension>
4247     </xsd:complexContent>
4248 </xsd:complexType>
4249
4250 <xsd:complexType name="tLocalTask">
4251     <xsd:complexContent>
4252         <xsd:extension base="tExtensibleElements">
4253             <xsd:sequence>
4254                 <xsd:element name="priority" type="tPriority-expr" minOccurs="0" />
4255                 <xsd:element name="peopleAssignments" type="tPeopleAssignments"
4256 minOccurs="0" />
4257             </xsd:sequence>
4258             <xsd:attribute name="reference" type="xsd:QName" use="required" />
4259         </xsd:extension>
4260     </xsd:complexContent>
4261 </xsd:complexType>
4262
4263 <!-- lean tasks -->
4264 <xsd:element name="leanTask" type="tLeanTask"/>
4265 <xsd:complexType name="tLeanTask">
4266     <xsd:complexContent>
4267         <xsd:restriction base="tTaskBase">
4268             <xsd:sequence>
4269                 <xsd:element name="documentation" type="tDocumentation"
4270 minOccurs="0" maxOccurs="unbounded" />
4271                 <xsd:any namespace="##other" processContents="lax" minOccurs="0"
4272 maxOccurs="unbounded" />
4273                 <xsd:element name="interface" type="tTaskInterface" minOccurs="0"
4274 maxOccurs="0" />
4275                 <xsd:element name="messageSchema" type="tMessageSchema" />

```

```

4276         <xsd:element name="priority" type="tPriority-expr" minOccurs="0" />
4277         <xsd:element name="peopleAssignments" type="tPeopleAssignments"
4278 minOccurs="0" />
4279         <xsd:element name="delegation" type="tDelegation" minOccurs="0" />
4280         <xsd:element name="presentationElements"
4281 type="tPresentationElements" minOccurs="0" />
4282         <xsd:element name="outcome" type="tQuery" minOccurs="0" />
4283         <xsd:element name="searchBy" type="tExpression" minOccurs="0" />
4284         <xsd:element name="renderings" type="tRenderings" minOccurs="0" />
4285         <xsd:element name="deadlines" type="tDeadlines" minOccurs="0" />
4286         <xsd:element name="composition" type="tComposition" minOccurs="0"
4287 maxOccurs="0" />
4288     </xsd:sequence>
4289     <xsd:attribute name="name" type="xsd:NCName" use="required" />
4290     <xsd:attribute name="actualOwnerRequired" type="tBoolean"
4291 use="optional" default="yes" />
4292     <xsd:anyAttribute namespace="##other" processContents="lax" />
4293 </xsd:restriction>
4294 </xsd:complexContent>
4295 </xsd:complexType>
4296
4297 <xsd:complexType name="tMessageSchema">
4298     <xsd:complexContent>
4299         <xsd:extension base="tExtensibleElements">
4300             <xsd:sequence>
4301                 <xsd:element name="messageField" type="tMessageField"
4302 minOccurs="0" maxOccurs="unbounded" />
4303             </xsd:sequence>
4304         </xsd:extension>
4305     </xsd:complexContent>
4306 </xsd:complexType>
4307
4308 <xsd:complexType name="tMessageField">
4309     <xsd:complexContent>
4310         <xsd:extension base="tExtensibleElements">
4311             <xsd:sequence>
4312                 <xsd:element name="messageDisplay" type="tMessageDisplay"
4313 maxOccurs="unbounded" />
4314                 <xsd:element name="messageChoice" type="tMessageChoice"
4315 minOccurs="0" maxOccurs="unbounded" />
4316             </xsd:sequence>
4317             <xsd:attribute name="name" type="xsd:NCName" />
4318             <xsd:attribute name="type" type="xsd:QName" />
4319         </xsd:extension>
4320     </xsd:complexContent>
4321 </xsd:complexType>
4322
4323 <xsd:complexType name="tMessageChoice">
4324     <xsd:complexContent>
4325         <xsd:extension base="tExtensibleElements">
4326             <xsd:sequence>
4327                 <xsd:element name="messageDisplay" type="tMessageDisplay"
4328 maxOccurs="unbounded" />
4329             </xsd:sequence>
4330             <xsd:attribute name="value" type="xsd:anySimpleType" />
4331         </xsd:extension>
4332     </xsd:complexContent>
4333 </xsd:complexType>

```

```

4334
4335 <xsd:complexType name="tMessageDisplay">
4336   <xsd:complexContent>
4337     <xsd:extension base="tExtensibleElements">
4338       <xsd:attribute ref="xml:lang" />
4339     </xsd:extension>
4340   </xsd:complexContent>
4341 </xsd:complexType>
4342
4343 <!-- notifications -->
4344 <xsd:element name="notifications" type="tNotifications" />
4345 <xsd:complexType name="tNotifications">
4346   <xsd:complexContent>
4347     <xsd:extension base="tExtensibleElements">
4348       <xsd:sequence>
4349         <xsd:element name="notification" type="tNotification"
4350 maxOccurs="unbounded" />
4351       </xsd:sequence>
4352     </xsd:extension>
4353   </xsd:complexContent>
4354 </xsd:complexType>
4355
4356 <xsd:element name="notification" type="tNotification" />
4357 <xsd:complexType name="tNotification">
4358   <xsd:complexContent>
4359     <xsd:extension base="tExtensibleElements">
4360       <xsd:sequence>
4361         <xsd:element name="interface" type="tNotificationInterface" />
4362         <xsd:element name="priority" type="tPriority-expr" minOccurs="0" />
4363         <xsd:element name="peopleAssignments" type="tPeopleAssignments" />
4364         <xsd:element name="presentationElements"
4365 type="tPresentationElements" />
4366         <xsd:element name="renderings" type="tRenderings" minOccurs="0" />
4367       </xsd:sequence>
4368       <xsd:attribute name="name" type="xsd:NCName" use="required" />
4369     </xsd:extension>
4370   </xsd:complexContent>
4371 </xsd:complexType>
4372
4373 <xsd:complexType name="tNotificationInterface">
4374   <xsd:complexContent>
4375     <xsd:extension base="tExtensibleElements">
4376       <xsd:attribute name="portType" type="xsd:QName" use="required" />
4377       <xsd:attribute name="operation" type="xsd:NCName" use="required" />
4378     </xsd:extension>
4379   </xsd:complexContent>
4380 </xsd:complexType>
4381
4382 <!-- miscellaneous helper types -->
4383 <xsd:complexType name="tText" mixed="true">
4384   <xsd:complexContent>
4385     <xsd:extension base="tExtensibleMixedContentElements">
4386       <xsd:attribute ref="xml:lang" />
4387     </xsd:extension>
4388   </xsd:complexContent>
4389 </xsd:complexType>
4390
4391 <xsd:complexType name="tDescription" mixed="true">

```

```

4392     <xsd:complexContent>
4393         <xsd:extension base="tExtensibleMixedContentElements">
4394             <xsd:attribute ref="xml:lang" />
4395             <xsd:attribute name="contentType" type="xsd:string" />
4396         </xsd:extension>
4397     </xsd:complexContent>
4398 </xsd:complexType>
4399
4400 <xsd:complexType name="tFrom" mixed="true">
4401     <xsd:complexContent>
4402         <xsd:extension base="tExtensibleMixedContentElements">
4403             <xsd:sequence>
4404                 <xsd:choice>
4405                     <xsd:element name="argument" type="tArgument" minOccurs="0"
4406 maxOccurs="unbounded" />
4407                     <xsd:element name="literal" type="tLiteral" minOccurs="0" />
4408                 </xsd:choice>
4409             </xsd:sequence>
4410             <xsd:attribute name="expressionLanguage" type="xsd:anyURI" />
4411             <xsd:attribute name="logicalPeopleGroup" type="xsd:NCName" />
4412         </xsd:extension>
4413     </xsd:complexContent>
4414 </xsd:complexType>
4415
4416 <xsd:complexType name="tArgument">
4417     <xsd:complexContent>
4418         <xsd:extension base="tExtensibleMixedContentElements">
4419             <xsd:attribute name="name" type="xsd:NCName" />
4420             <xsd:attribute name="expressionLanguage" type="xsd:anyURI" />
4421         </xsd:extension>
4422     </xsd:complexContent>
4423 </xsd:complexType>
4424
4425 <xsd:complexType name="tParameter" mixed="true">
4426     <xsd:complexContent>
4427         <xsd:extension base="tExtensibleMixedContentElements">
4428             <xsd:attribute name="name" type="xsd:NCName" use="required" />
4429             <xsd:attribute name="type" type="xsd:QName" use="required" />
4430         </xsd:extension>
4431     </xsd:complexContent>
4432 </xsd:complexType>
4433
4434 <xsd:complexType name="tLiteral" mixed="true">
4435     <xsd:sequence>
4436         <xsd:any namespace="##any" processContents="lax" />
4437     </xsd:sequence>
4438     <xsd:anyAttribute namespace="##other" processContents="lax" />
4439 </xsd:complexType>
4440
4441 <xsd:complexType name="tQuery" mixed="true">
4442     <xsd:complexContent>
4443         <xsd:extension base="tExtensibleMixedContentElements">
4444             <xsd:attribute name="part" />
4445             <xsd:attribute name="queryLanguage" type="xsd:anyURI" />
4446         </xsd:extension>
4447     </xsd:complexContent>
4448 </xsd:complexType>
4449

```

```

4450 <xsd:complexType name="tExpression" mixed="true">
4451   <xsd:complexContent>
4452     <xsd:extension base="tExtensibleMixedContentElements">
4453       <xsd:attribute name="expressionLanguage" type="xsd:anyURI" />
4454     </xsd:extension>
4455   </xsd:complexContent>
4456 </xsd:complexType>
4457
4458 <xsd:element name="priority" type="tPriority-expr" />
4459 <xsd:complexType name="tPriority-expr" mixed="true">
4460   <xsd:complexContent mixed="true">
4461     <xsd:extension base="tExpression" />
4462   </xsd:complexContent>
4463 </xsd:complexType>
4464
4465 <xsd:complexType name="tBoolean-expr" mixed="true">
4466   <xsd:complexContent mixed="true">
4467     <xsd:extension base="tExpression" />
4468   </xsd:complexContent>
4469 </xsd:complexType>
4470
4471 <xsd:complexType name="tDuration-expr" mixed="true">
4472   <xsd:complexContent mixed="true">
4473     <xsd:extension base="tExpression" />
4474   </xsd:complexContent>
4475 </xsd:complexType>
4476
4477 <xsd:complexType name="tDeadline-expr" mixed="true">
4478   <xsd:complexContent mixed="true">
4479     <xsd:extension base="tExpression" />
4480   </xsd:complexContent>
4481 </xsd:complexType>
4482
4483 <xsd:simpleType name="tBoolean">
4484   <xsd:restriction base="xsd:string">
4485     <xsd:enumeration value="yes" />
4486     <xsd:enumeration value="no" />
4487   </xsd:restriction>
4488 </xsd:simpleType>
4489
4490 </xsd:schema>

```


C. WS-HumanTask Data Types Schema

```

4492 | -<?xml version="1.0" encoding="UTF-8"?>
4493 | <!--
4494 |   Copyright (c) OASIS Open 2009. All Rights Reserved.
4495 | -->
4496 | <xsd:schema
4497 |   targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
4498 | humantask/types/200803"
4499 |   xmlns="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/types/200803"
4500 |   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
4501 |   elementFormDefault="qualified"
4502 |   blockDefault="#all">
4503 |
4504 |   <xsd:annotation>
4505 |     <xsd:documentation>
4506 |       XML Schema for WS-HumanTask 1.1 - WS-HumanTask Data Type Definitions
4507 |     </xsd:documentation>
4508 |   </xsd:annotation>
4509 |
4510 |   <!-- other namespaces -->
4511 |   <xsd:import namespace="http://www.w3.org/XML/1998/namespace"
4512 | schemaLocation="http://www.w3.org/2001/xml.xsd" />
4513 |
4514 |   <!-- data types for attachment operations -->
4515 |   <xsd:element name="attachmentInfo" type="tAttachmentInfo" />
4516 |   <xsd:complexType name="tAttachmentInfo">
4517 |     <xsd:sequence>
4518 |       <xsd:element name="identifier" type="xsd:anyURI" />
4519 |       <xsd:element name="name" type="xsd:string" />
4520 |       <xsd:element name="accessType" type="xsd:string" />
4521 |       <xsd:element name="contentType" type="xsd:string" />
4522 |       <xsd:element name="contentCategory" type="xsd:anyURI" />
4523 |       <xsd:element name="attachedTime" type="xsd:dateTime" />
4524 |       <xsd:element name="attachedBy" type="tUser" />
4525 |       <xsd:any namespace="##other" processContents="lax" minOccurs="0"
4526 | maxOccurs="unbounded" />
4527 |     </xsd:sequence>
4528 |   </xsd:complexType>
4529 |   <xsd:element name="attachment" type="tAttachment" />
4530 |   <xsd:complexType name="tAttachment">
4531 |     <xsd:sequence>
4532 |       <xsd:element ref="attachmentInfo" />
4533 |       <xsd:element name="value" type="xsd:anyType" />
4534 |     </xsd:sequence>
4535 |   </xsd:complexType>
4536 |
4537 |   <!-- data types for comments -->
4538 |   <xsd:element name="comment" type="tComment" />
4539 |   <xsd:complexType name="tComment">
4540 |     <xsd:sequence>
4541 |       <xsd:element name="id" type="xsd:anyURI" />
4542 |       <xsd:element name="addedTime" type="xsd:dateTime" />
4543 |       <xsd:element name="addedBy" type="tUser" />
4544 |       <xsd:element name="lastModifiedTime" type="xsd:dateTime" />

```



```

4545 | <xsd:element name="lastModifiedBy" type="tUser"/>
4546 | <xsd:element name="text" type="xsd:string"/>
4547 | <xsd:any namespace="##other" processContents="lax" minOccurs="0"
4548 | maxOccurs="unbounded"/>
4549 | </xsd:sequence>
4550 | </xsd:complexType>
4551 |
4552 | <!-- data types for simple query operations -->
4553 | <xsd:element name="taskAbstract" type="tTaskAbstract"/>
4554 | <xsd:complexType name="tTaskAbstract">
4555 | <xsd:sequence>
4556 | <xsd:element name="id" type="xsd:stringanyURI"/>
4557 | <xsd:element name="taskType" type="xsd:string"/>
4558 | <xsd:element name="name" type="xsd:QName"/>
4559 | <xsd:element name="status" type="tStatus"/>
4560 | <xsd:element name="priority" type="tPriority" minOccurs="0"/>
4561 | <xsd:element name="createdTime" type="xsd:dateTime"/>
4562 | <xsd:element name="activationTime" type="xsd:dateTime" minOccurs="0"/>
4563 | <xsd:element name="expirationTime" type="xsd:dateTime" minOccurs="0"/>
4564 | <xsd:element name="isSkipable" type="xsd:boolean" minOccurs="0"/>
4565 | <xsd:element name="hasPotentialOwners" type="xsd:boolean"
4566 | minOccurs="0"/>
4567 | <xsd:element name="startByTimeExists" type="xsd:boolean"
4568 | minOccurs="0"/>
4569 | <xsd:element name="completeByTimeExists" type="xsd:boolean"
4570 | minOccurs="0"/>
4571 | <xsd:element name="presentationName" type="tPresentationName"
4572 | minOccurs="0"/>
4573 | <xsd:element name="presentationSubject" type="tPresentationSubject"
4574 | minOccurs="0"/>
4575 | <xsd:element name="renderingMethodExists" type="xsd:boolean"/>
4576 | <xsd:element name="hasOutput" type="xsd:boolean" minOccurs="0"/>
4577 | <xsd:element name="hasFault" type="xsd:boolean" minOccurs="0"/>
4578 | <xsd:element name="hasAttachments" type="xsd:boolean" minOccurs="0"/>
4579 | <xsd:element name="hasComments" type="xsd:boolean" minOccurs="0"/>
4580 | <xsd:element name="escalated" type="xsd:boolean" minOccurs="0"/>
4581 | <xsd:element name="outcome" type="xsd:string" minOccurs="0"/>
4582 | <xsd:element name="parentTaskId" type="xsd:stringanyURI"
4583 | minOccurs="0"/>
4584 | <xsd:element name="hasSubTasks" type="xsd:boolean" minOccurs="0"/>
4585 | <xsd:any namespace="##other" processContents="lax" minOccurs="0"
4586 | maxOccurs="unbounded"/>
4587 | </xsd:sequence>
4588 | </xsd:complexType>
4589 | <xsd:element name="taskDetails" type="tTaskDetails"/>
4590 | <xsd:complexType name="tTaskDetails">
4591 | <xsd:sequence>
4592 | <xsd:element name="id" type="xsd:stringanyURI"/>
4593 | <xsd:element name="taskType" type="xsd:string"/>
4594 | <xsd:element name="name" type="xsd:QName"/>
4595 | <xsd:element name="status" type="tStatus"/>
4596 | <xsd:element name="priority" type="tPriority" minOccurs="0"/>
4597 | <xsd:element name="taskInitiator" type="tUser" minOccurs="0"/>
4598 | <xsd:element name="taskStakeholders" type="tOrganizationalEntity"
4599 | minOccurs="0"/>
4600 | <xsd:element name="potentialOwners" type="tOrganizationalEntity"
4601 | minOccurs="0"/>

```

```

4602     <xsd:element name="businessAdministrators" type="tOrganizationalEntity"
4603 minOccurs="0"/>
4604     <xsd:element name="actualOwner" type="tUser" minOccurs="0"/>
4605     <xsd:element name="notificationRecipients" type="tOrganizationalEntity"
4606 minOccurs="0"/>
4607     <xsd:element name="createdTime" type="xsd:dateTime"/>
4608     <xsd:element name="createdBy" type="xsd:string" minOccurs="0"/>
4609     <xsd:element name="lastModifiedTime" type="xsd:dateTime"/>
4610     <xsd:element name="lastModifiedBy" type="xsd:string" minOccurs="0"/>
4611     <xsd:element name="activationTime" type="xsd:dateTime" minOccurs="0"/>
4612     <xsd:element name="expirationTime" type="xsd:dateTime" minOccurs="0"/>
4613     <xsd:element name="isSkipable" type="xsd:boolean" minOccurs="0"/>
4614     <xsd:element name="hasPotentialOwners" type="xsd:boolean"
4615 minOccurs="0"/>
4616     <xsd:element name="startByTimeExists" type="xsd:boolean"
4617 minOccurs="0"/>
4618     <xsd:element name="completeByTimeExists" type="xsd:boolean"
4619 minOccurs="0"/>
4620     <xsd:element name="presentationName" type="tPresentationName"
4621 minOccurs="0"/>
4622     <xsd:element name="presentationSubject" type="tPresentationSubject"
4623 minOccurs="0"/>
4624     <xsd:element name="renderingMethodExists" type="xsd:boolean"/>
4625     <xsd:element name="hasOutput" type="xsd:boolean" minOccurs="0"/>
4626     <xsd:element name="hasFault" type="xsd:boolean" minOccurs="0"/>
4627     <xsd:element name="hasAttachments" type="xsd:boolean" minOccurs="0"/>
4628     <xsd:element name="hasComments" type="xsd:boolean" minOccurs="0"/>
4629     <xsd:element name="escalated" type="xsd:boolean" minOccurs="0"/>
4630     <xsd:element name="searchBy" type="xsd:string" minOccurs="0"/>
4631     <xsd:element name="outcome" type="xsd:string" minOccurs="0"/>
4632     <xsd:element name="parentTaskId" type="xsd:stringanyURI"
4633 minOccurs="0"/>
4634     <xsd:element name="hasSubTasks" type="xsd:boolean" minOccurs="0"/>
4635     <xsd:any namespace="##other" processContents="lax" minOccurs="0"
4636 maxOccurs="unbounded"/>
4637 </xsd:sequence>
4638 </xsd:complexType>
4639 <xsd:simpleType name="tPresentationName">
4640 <xsd:annotation>
4641 <xsd:documentation>length-restricted string</xsd:documentation>
4642 </xsd:annotation>
4643 <xsd:restriction base="xsd:string">
4644 <xsd:maxLength value="64"/>
4645 <xsd:whiteSpace value="preserve"/>
4646 </xsd:restriction>
4647 </xsd:simpleType>
4648 <xsd:simpleType name="tPresentationSubject">
4649 <xsd:annotation>
4650 <xsd:documentation>length-restricted string</xsd:documentation>
4651 </xsd:annotation>
4652 <xsd:restriction base="xsd:string">
4653 <xsd:maxLength value="254"/>
4654 <xsd:whiteSpace value="preserve"/>
4655 </xsd:restriction>
4656 </xsd:simpleType>
4657 <xsd:simpleType name="tStatus">
4658 <xsd:restriction base="xsd:string"/>
4659 </xsd:simpleType>

```

```

4660 <xsd:simpleType name="tPredefinedStatus">
4661   <xsd:annotation>
4662     <xsd:documentation>for documentation only</xsd:documentation>
4663   </xsd:annotation>
4664   <xsd:restriction base="xsd:string">
4665     <xsd:enumeration value="CREATED" />
4666     <xsd:enumeration value="READY" />
4667     <xsd:enumeration value="RESERVED" />
4668     <xsd:enumeration value="IN_PROGRESS" />
4669     <xsd:enumeration value="SUSPENDED" />
4670     <xsd:enumeration value="COMPLETED" />
4671     <xsd:enumeration value="FAILED" />
4672     <xsd:enumeration value="ERROR" />
4673     <xsd:enumeration value="EXITED" />
4674     <xsd:enumeration value="OBSOLETE" />
4675   </xsd:restriction>
4676 </xsd:simpleType>
4677 <xsd:simpleType name="tPriority">
4678   <xsd:restriction base="xsd:integer">
4679     <xsd:minInclusive value="0" />
4680     <xsd:maxInclusive value="10" />
4681   </xsd:restriction>
4682 </xsd:simpleType>
4683 <xsd:complexType name="tTime">
4684   <xsd:choice>
4685     <xsd:element name="timePeriod" type="xsd:duration" />
4686     <xsd:element name="pointOfTime" type="xsd:dateTime" />
4687   </xsd:choice>
4688 </xsd:complexType>
4689
4690 <!-- task operations -->
4691 <xsd:complexType name="tTaskOperations">
4692   <xsd:choice maxOccurs="unbounded">
4693     <xsd:element name="activate" type="tTaskOperation" />
4694     <xsd:element name="addAttachment" type="tTaskOperation" />
4695     <xsd:element name="addComment" type="tTaskOperation" />
4696     <xsd:element name="claim" type="tTaskOperation" />
4697     <xsd:element name="complete" type="tTaskOperation" />
4698     <xsd:element name="delegate" type="tTaskOperation" />
4699     <xsd:element name="deleteAttachment" type="tTaskOperation" />
4700     <xsd:element name="deleteComment" type="tTaskOperation" />
4701     <xsd:element name="deleteFault" type="tTaskOperation" />
4702     <xsd:element name="deleteOutput" type="tTaskOperation" />
4703     <xsd:element name="fail" type="tTaskOperation" />
4704     <xsd:element name="forward" type="tTaskOperation" />
4705     <xsd:element name="getAttachment" type="tTaskOperation" />
4706     <xsd:element name="getAttachmentInfos" type="tTaskOperation" />
4707     <xsd:element name="getComments" type="tTaskOperation" />
4708     <xsd:element name="getFault" type="tTaskOperation" />
4709     <xsd:element name="getInput" type="tTaskOperation" />
4710     <xsd:element name="getOutcome" type="tTaskOperation" />
4711     <xsd:element name="getOutput" type="tTaskOperation" />
4712     <xsd:element name="getParentTask" type="tTaskOperation" />
4713     <xsd:element name="getParentTaskIdentifier" type="tTaskOperation" />
4714     <xsd:element name="getRendering" type="tTaskOperation" />
4715     <xsd:element name="getRenderingTypes" type="tTaskOperation" />
4716     <xsd:element name="getSubtaskIdentifiers" type="tTaskOperation" />
4717     <xsd:element name="getSubtasks" type="tTaskOperation" />

```

4718 <xsd:element name="getTaskDescription" type="tTaskOperation"/>
4719 <xsd:element name="getTaskDetails" type="tTaskOperation"/>
4720 <xsd:element name="getTaskHistory" type="tTaskOperation"/>
4721 <xsd:element name="getTaskInstanceData" type="tTaskOperation"/>
4722 <xsd:element name="hasSubtasks" type="tTaskOperation"/>
4723 <xsd:element name="instantiateSubtask" type="tTaskOperation"/>
4724 <xsd:element name="isSubtask" type="tTaskOperation"/>
4725 <xsd:element name="nominate" type="tTaskOperation"/>
4726 <xsd:element name="release" type="tTaskOperation"/>
4727 <xsd:element name="remove" type="tTaskOperation"/>
4728 <xsd:element name="resume" type="tTaskOperation"/>
4729 <xsd:element name="setFault" type="tTaskOperation"/>
4730 <xsd:element name="setGenericHumanRole" type="tTaskOperation"/>
4731 <xsd:element name="setOutput" type="tTaskOperation"/>
4732 <xsd:element name="setPriority" type="tTaskOperation"/>
4733 <xsd:element name="setTaskCompletionDeadlineExpression"
4734 type="tTaskOperation"/>
4735 <xsd:element name="setTaskCompletionDurationExpression"
4736 type="tTaskOperation"/>
4737 <xsd:element name="setTaskStartDeadlineExpression"
4738 type="tTaskOperation"/>
4739 <xsd:element name="setTaskStartDurationExpression"
4740 type="tTaskOperation"/>
4741 <xsd:element name="skip" type="tTaskOperation"/>
4742 <xsd:element name="start" type="tTaskOperation"/>
4743 <xsd:element name="stop" type="tTaskOperation"/>
4744 ~~<xsd:element name="release" type="tTaskOperation"/>~~
4745 <xsd:element name="suspend" type="tTaskOperation"/>
4746 <xsd:element name="suspendUntil" type="tTaskOperation"/>
4747 ~~<xsd:element name="resume" type="tTaskOperation"/>~~
4748 ~~<xsd:element name="complete" type="tTaskOperation"/>~~
4749 ~~<xsd:element name="remove" type="tTaskOperation"/>~~
4750 ~~<xsd:element name="fail" type="tTaskOperation"/>~~
4751 ~~<xsd:element name="setPriority" type="tTaskOperation"/>~~
4752 ~~<xsd:element name="addAttachment" type="tTaskOperation"/>~~
4753 ~~<xsd:element name="getAttachmentInfos" type="tTaskOperation"/>~~
4754 ~~<xsd:element name="getAttachment" type="tTaskOperation"/>~~
4755 ~~<xsd:element name="deleteAttachment" type="tTaskOperation"/>~~
4756 ~~<xsd:element name="addComment" type="tTaskOperation"/>~~
4757 <xsd:element name="updateComment" type="tTaskOperation"/>
4758 ~~<xsd:element name="deleteComment" type="tTaskOperation"/>~~
4759 ~~<xsd:element name="getComments" type="tTaskOperation"/>~~
4760 ~~<xsd:element name="skip" type="tTaskOperation"/>~~
4761 ~~<xsd:element name="forward" type="tTaskOperation"/>~~
4762 ~~<xsd:element name="delegate" type="tTaskOperation"/>~~
4763 ~~<xsd:element name="getRendering" type="tTaskOperation"/>~~
4764 ~~<xsd:element name="getRenderingTypes" type="tTaskOperation"/>~~
4765 ~~<xsd:element name="getTaskDetails" type="tTaskOperation"/>~~
4766 ~~<xsd:element name="getTaskDescription" type="tTaskOperation"/>~~
4767 ~~<xsd:element name="setOutput" type="tTaskOperation"/>~~
4768 ~~<xsd:element name="deleteOutput" type="tTaskOperation"/>~~
4769 ~~<xsd:element name="setFault" type="tTaskOperation"/>~~
4770 ~~<xsd:element name="deleteFault" type="tTaskOperation"/>~~
4771 ~~<xsd:element name="getInput" type="tTaskOperation"/>~~
4772 ~~<xsd:element name="getOutput" type="tTaskOperation"/>~~
4773 ~~<xsd:element name="getFault" type="tTaskOperation"/>~~
4774 ~~<xsd:element name="getOutcome" type="tTaskOperation"/>~~
4775 ~~<xsd:element name="getTaskHistory" type="tTaskOperation"/>~~

```

4776 <xsd:element name="getTaskInstanceData" type="tTaskOperation"/>
4777 <xsd:element name="getSubtasks" type="tTaskOperation"/>
4778 <xsd:element name="getSubtaskIdentifiers" type="tTaskOperation"/>
4779 <xsd:element name="hasSubtasks" type="tTaskOperation"/>
4780 <xsd:element name="getParentTask" type="tTaskOperation"/>
4781 <xsd:element name="getParentTaskIdentifier" type="tTaskOperation"/>
4782 <xsd:element name="isSubtask" type="tTaskOperation"/>
4783 <xsd:element name="instantiateSubtask" type="tTaskOperation"/>
4784 <xsd:element name="setTaskStartDeadlineExpression"
4785 type="tTaskOperation"/>
4786 <xsd:element name="setTaskStartDurationExpression"
4787 type="tTaskOperation"/>
4788 <xsd:element name="setTaskCompletionDeadlineExpression"
4789 type="tTaskOperation"/>
4790 <xsd:element name="setTaskCompletionDurationExpression"
4791 type="tTaskOperation"/>
4792 <xsd:element name="activate" type="tTaskOperation"/>
4793 <xsd:element name="nominate" type="tTaskOperation"/>
4794 <xsd:element name="setGenericHumanRole" type="tTaskOperation"/>
4795 <xsd:any namespace="##other" processContents="lax"/>
4796 </xsd:choice>
4797 </xsd:complexType>
4798 <xsd:complexType name="tTaskOperation">
4799 <xsd:complexContent>
4800 <xsd:restriction base="xsd:anyType"/>
4801 </xsd:complexContent>
4802 </xsd:complexType>
4803
4804 <!-- data types for advanced query operations -->
4805 <xsd:element name="taskQueryResultSet" type="tTaskQueryResultSet"/>
4806 <xsd:complexType name="tTaskQueryResultSet">
4807 <xsd:sequence>
4808 <xsd:element name="row" type="tTaskQueryResultRow" minOccurs="0"
4809 maxOccurs="unbounded"/>
4810 </xsd:sequence>
4811 </xsd:complexType>
4812 <xsd:complexType name="tTaskQueryResultRow">
4813 <xsd:choice minOccurs="0" maxOccurs="unbounded">
4814 <xsd:element name="id" type="xsd:stringanyURI"/>
4815 <xsd:element name="taskType" type="xsd:string"/>
4816 <xsd:element name="name" type="xsd:QName"/>
4817 <xsd:element name="status" type="tStatus"/>
4818 <xsd:element name="priority" type="tPriority"/>
4819 <xsd:element name="taskInitiator" type="tOrganizationalEntity"/>
4820 <xsd:element name="taskStakeholders" type="tOrganizationalEntity"/>
4821 <xsd:element name="potentialOwners" type="tOrganizationalEntity"/>
4822 <xsd:element name="businessAdministrators"
4823 type="tOrganizationalEntity"/>
4824 <xsd:element name="actualOwner" type="tUser"/>
4825 <xsd:element name="notificationRecipients"
4826 type="tOrganizationalEntity"/>
4827 <xsd:element name="createdTime" type="xsd:dateTime"/>
4828 <xsd:element name="createdBy" type="xsd:string"/>
4829 <xsd:element name="lastModifiedTime" type="xsd:dateTime"/>
4830 <xsd:element name="lastModifiedBy" type="xsd:string"/>
4831 <xsd:element name="activationTime" type="xsd:dateTime"/>
4832 <xsd:element name="expirationTime" type="xsd:dateTime"/>
4833 <xsd:element name="isSkipable" type="xsd:boolean"/>

```



```

4834     <xsd:element name="hasPotentialOwners" type="xsd:boolean" />
4835     <xsd:element name="startByTime" type="xsd:dateTime" />
4836     <xsd:element name="completeByTime" type="xsd:dateTime" />
4837     <xsd:element name="presentationName" type="tPresentationName" />
4838     <xsd:element name="presentationSubject" type="tPresentationSubject" />
4839     <xsd:element name="renderingMethodName" type="xsd:QName" />
4840     <xsd:element name="hasOutput" type="xsd:boolean" />
4841     <xsd:element name="hasFault" type="xsd:boolean" />
4842     <xsd:element name="hasAttachments" type="xsd:boolean" />
4843     <xsd:element name="hasComments" type="xsd:boolean" />
4844     <xsd:element name="escalated" type="xsd:boolean" />
4845     <xsd:element name="parentTaskId" type="xsd:stringanyURI" />
4846     <xsd:element name="hasSubtasks" type="xsd:boolean" />
4847     <xsd:element name="searchBy" type="xsd:string" />
4848     <xsd:element name="outcome" type="xsd:string" />
4849     <xsd:element name="taskOperations" type="tTaskOperations" />
4850     <xsd:any namespace="##other" processContents="lax" />
4851   </xsd:choice>
4852 </xsd:complexType>
4853 <xsd:complexType name="tFault">
4854   <xsd:sequence>
4855     <xsd:element name="faultName" type="xsd:NCName" />
4856     <xsd:element name="faultData" type="xsd:anyType" />
4857   </xsd:sequence>
4858 </xsd:complexType>
4859
4860 <!-- elements and types for organizational entities -->
4861 <xsd:element name="organizationalEntity" type="tOrganizationalEntity" />
4862 <xsd:complexType name="tOrganizationalEntity">
4863   <xsd:choice maxOccurs="unbounded">
4864     <xsd:element name="user" type="tUser" />
4865     <xsd:element name="group" type="tGroup" />
4866   </xsd:choice>
4867 </xsd:complexType>
4868 <xsd:element name="user" type="tUser" />
4869 <xsd:simpleType name="tUser">
4870   <xsd:restriction base="xsd:string" />
4871 </xsd:simpleType>
4872 <xsd:element name="group" type="tGroup" />
4873 <xsd:simpleType name="tGroup">
4874   <xsd:restriction base="xsd:string" />
4875 </xsd:simpleType>
4876
4877 <!-- input or output message part data -->
4878 <xsd:element name="part" type="tPart" />
4879 <xsd:complexType name="tPart" mixed="true">
4880   <xsd:sequence>
4881     <xsd:any processContents="skip" minOccurs="0" />
4882   </xsd:sequence>
4883   <xsd:attribute name="name" type="xsd:NCName" use="required" />
4884 </xsd:complexType>
4885
4886 <!-- type container element for one or more message parts -->
4887 <xsd:complexType name="tMessagePartsData">
4888   <xsd:sequence>
4889     <xsd:element ref="part" minOccurs="0" maxOccurs="unbounded" />
4890   </xsd:sequence>
4891 </xsd:complexType>

```

```

4892 <xsd:complexType name="tFaultData">
4893   <xsd:sequence>
4894     <xsd:element name="faultName" type="xsd:NCName"/>
4895     <xsd:element name="faultData" type="xsd:anyType"/>
4896   </xsd:sequence>
4897 </xsd:complexType>
4898 <xsd:element name="attachmentInfos" type="tAttachmentInfos"/>
4899 <xsd:complexType name="tAttachmentInfos">
4900   <xsd:sequence>
4901     <xsd:element name="info" type="tAttachmentInfo" minOccurs="0"
4902 maxOccurs="unbounded"/>
4903   </xsd:sequence>
4904 </xsd:complexType>
4905 <xsd:element name="comments" type="tComments"/>
4906 <xsd:complexType name="tComments">
4907   <xsd:sequence>
4908     <xsd:element ref="comment" minOccurs="0" maxOccurs="unbounded"/>
4909   </xsd:sequence>
4910 </xsd:complexType>
4911 <xsd:element name="renderingType" type="xsd:QName"/>
4912 <xsd:complexType name="tRenderingTypes">
4913   <xsd:sequence>
4914     <xsd:element ref="renderingType" minOccurs="0" maxOccurs="unbounded"/>
4915   </xsd:sequence>
4916 </xsd:complexType>
4917
4918 <!-- Single rendering element that contains rendering type (attribute) and
4919 data. -->
4920 <xsd:element name="rendering" type="tRendering"/>
4921 <xsd:complexType name="tRendering">
4922   <xsd:sequence>
4923     <xsd:any namespace="##other" processContents="lax" minOccurs="0"
4924 maxOccurs="unbounded"/>
4925   </xsd:sequence>
4926   <xsd:attribute name="type" type="xsd:QName" use="required"/>
4927 </xsd:complexType>
4928 <xsd:element name="renderings">
4929   <xsd:complexType>
4930     <xsd:sequence>
4931       <xsd:element ref="rendering" minOccurs="0" maxOccurs="unbounded"/>
4932     </xsd:sequence>
4933   </xsd:complexType>
4934 </xsd:element>
4935 <xsd:element name="description" type="xsd:string"/>
4936 <xsd:complexType name="tTaskInstanceData">
4937   <xsd:sequence>
4938     <!-- taskDetails contains task ID, meta data, presentation name and
4939 presentation subject. -->
4940     <xsd:element ref="taskDetails"/>
4941     <xsd:element ref="description"/>
4942     <xsd:element name="input" type="tMessagePartsData"/>
4943     <xsd:element name="output" type="tMessagePartsData" nillable="true"/>
4944     <xsd:element name="fault" type="tFaultData" nillable="true"
4945 minOccurs="0"/>
4946     <xsd:element ref="renderings" minOccurs="0"/>
4947     <xsd:element ref="comments" minOccurs="0"/>
4948     <xsd:element ref="attachmentInfos" minOccurs="0"/>

```

```

4949     <xsd:any namespace="##other" processContents="lax" minOccurs="0"
4950 maxOccurs="unbounded" />
4951   </xsd:sequence>
4952 </xsd:complexType>
4953
4954 <!-- Defines the human task event types -->
4955 <xsd:simpleType name="tTaskEventType">
4956   <xsd:restriction base="xsd:string">
4957     <xsd:enumeration value="create" />
4958     <xsd:enumeration value="claim" />
4959     <xsd:enumeration value="start" />
4960     <xsd:enumeration value="stop" />
4961     <xsd:enumeration value="release" />
4962     <xsd:enumeration value="suspend" />
4963     <xsd:enumeration value="suspendUntil" />
4964     <xsd:enumeration value="resume" />
4965     <xsd:enumeration value="complete" />
4966     <xsd:enumeration value="remove" />
4967     <xsd:enumeration value="fail" />
4968     <xsd:enumeration value="setPriority" />
4969     <xsd:enumeration value="addAttachment" />
4970     <xsd:enumeration value="deleteattachment" />
4971     <xsd:enumeration value="addComment" />
4972     <xsd:enumeration value="skip" />
4973     <xsd:enumeration value="forward" />
4974     <xsd:enumeration value="delegate" />
4975     <xsd:enumeration value="setOutput" />
4976     <xsd:enumeration value="deleteOutput" />
4977     <xsd:enumeration value="setFault" />
4978     <xsd:enumeration value="deleteFault" />
4979     <xsd:enumeration value="activate" />
4980     <xsd:enumeration value="nominate" />
4981     <xsd:enumeration value="setGenericHumanRole" />
4982     <xsd:enumeration value="expire" />
4983     <xsd:enumeration value="escalated" />
4984   </xsd:restriction>
4985 </xsd:simpleType>
4986 <xsd:element name="taskEvent">
4987   <xsd:complexType>
4988     <xsd:annotation>
4989       <xsd:documentation>
4990         A detailed event that represents a change in the task's state.
4991       </xsd:documentation>
4992     </xsd:annotation>
4993   <xsd:sequence>
4994     <!-- event id - unique per task -->
4995     <xsd:element name="id" type="xsd:integer" />
4996     <!-- event date time -->
4997     <xsd:element name="eventTime" type="xsd:dateTime" />
4998     <!-- task ID -->
4999     <xsd:element name="identifier" type="xsd:anyURI" />
5000     <xsd:element name="principal" type="xsd:string" nillable="true"
5001 minOccurs="0" />
5002     <!-- Event type. Note - using a restricted type limits extensibility
5003 to add custom event types. -->
5004     <xsd:element name="eventType" type="tTaskEventType" />
5005     <!-- actual owner of the task before the event -->

```



```

5006     <xsd:element name="startOwner" type="xsd:string" nillable="true"
5007 minOccurs="0"/>
5008     <!-- actual owner of the task after the event -->
5009     <xsd:element name="endOwner" type="xsd:string" nillable="true"
5010 minOccurs="0"/>
5011     <!-- WSHHT task status -->
5012     <xsd:element name="status" type="tStatus"/>
5013     <!-- boolean to indicate this event has optional data -->
5014     <xsd:element name="hasData" type="xsd:boolean" minOccurs="0"/>
5015     <xsd:element name="eventData" type="xsd:anyType" nillable="true"
5016 minOccurs="0"/>
5017     <xsd:element name="faultName" type="xsd:string" nillable="true"
5018 minOccurs="0"/>
5019     <!-- extensibility -->
5020     <xsd:any namespace="##other" processContents="lax" minOccurs="0"
5021 maxOccurs="unbounded"/>
5022     </xsd:sequence>
5023     </xsd:complexType>
5024     </xsd:element>
5025     <!-- Filter allow list event by eventId or other params such as status and
5026 event type -->
5027     <xsd:complexType name="tTaskHistoryFilter">
5028     <xsd:choice>
5029     <xsd:element name="eventId" type="xsd:integer"/>
5030     <!-- Filter to allow narrow down query by status, principal, event
5031 Type. -->
5032     <xsd:sequence>
5033     <xsd:element name="status" type="tStatus" minOccurs="0"
5034 maxOccurs="unbounded"/>
5035     <xsd:element name="eventType" type="tTaskEventType" minOccurs="0"
5036 maxOccurs="unbounded"/>
5037     <xsd:element name="principal" type="xsd:string" minOccurs="0"/>
5038     <xsd:element name="afterEventTime" type="xsd:dateTime"
5039 minOccurs="0"/>
5040     <xsd:element name="beforeEventTime" type="xsd:dateTime"
5041 minOccurs="0"/>
5042     </xsd:sequence>
5043     </xsd:choice>
5044     </xsd:complexType>
5045 </xsd:schema>

```

5046

D. WS-HumanTask Client API Port Type

```
5047 <?xml version="1.0" encoding="UTF-8"?>
5048 <!--
5049 Copyright (c) OASIS Open 2009. All Rights Reserved.
5050 -->
5051 <wsdl:definitions
5052   targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
5053 humantask/api/200803"
5054   xmlns="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/api/200803"
5055   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
5056   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
5057   xmlns:htt="http://docs.oasis-open.org/ns/bpel4people/ws-
5058 humantask/types/200803">
5059
5060   <wsdl:documentation>
5061     Web Service Definition for WS-HumanTask 1.1 - Operations for Client
5062 Applications
5063   </wsdl:documentation>
5064
5065   <wsdl:types>
5066     <xsd:schema
5067       targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
5068 humantask/api/200803"
5069       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
5070       xmlns:htt="http://docs.oasis-open.org/ns/bpel4people/ws-
5071 humantask/types/200803"
5072       elementFormDefault="qualified"
5073       blockDefault="#all">
5074
5075       <xsd:import
5076         namespace="http://docs.oasis-open.org/ns/bpel4people/ws-
5077 humantask/types/200803"
5078         schemaLocation="ws-humantask-types.xsd"/>
5079
5080       <!-- Input and output elements -->
5081       <xsd:element name="addAttachment">
5082         <xsd:complexType>
5083           <xsd:sequence>
5084             <xsd:element name="taskIdIdentifier" type="xsd:anyURI"/>
5085             <xsd:element name="name" type="xsd:string"/>
5086             <xsd:element name="accessType" type="xsd:string"/>
5087             <xsd:element name="contentType" type="xsd:string"/>
5088             <xsd:element name="attachment" type="xsd:anyType"/>
5089           </xsd:sequence>
5090         </xsd:complexType>
5091       </xsd:element>
5092       <xsd:element name="addAttachmentResponse">
5093         <xsd:complexType>
5094           <xsd:sequence>
5095             <xsd:element name="identifier" type="xsd:anyURI"/>
5096           </xsd:sequence>
5097         </xsd:complexType>
5098       </xsd:element>
5099
```

```

5100     <xsd:element name="addComment">
5101         <xsd:complexType>
5102             <xsd:sequence>
5103                 <xsd:element name="identifier" type="xsd:anyURI"/>
5104                 <xsd:element name="text" type="xsd:string"/>
5105             </xsd:sequence>
5106         </xsd:complexType>
5107     </xsd:element>
5108     <xsd:element name="addCommentResponse">
5109         <xsd:complexType>
5110             <xsd:sequence>
5111                 <xsd:element name="commentID" type="xsd:anyURI"/>
5112             </xsd:sequence>
5113         </xsd:complexType>
5114     </xsd:element>
5115
5116     <xsd:element name="claim">
5117         <xsd:complexType>
5118             <xsd:sequence>
5119                 <xsd:element name="identifier" type="xsd:anyURI"/>
5120             </xsd:sequence>
5121         </xsd:complexType>
5122     </xsd:element>
5123     <xsd:element name="claimResponse">
5124         <xsd:complexType>
5125             <xsd:sequence>
5126                 <xsd:annotation>
5127                     <xsd:documentation>Empty message</xsd:documentation>
5128                 </xsd:annotation>
5129             </xsd:sequence>
5130         </xsd:complexType>
5131     </xsd:element>
5132
5133     <xsd:element name="batchClaim">
5134         <xsd:complexType>
5135             <xsd:sequence>
5136                 <xsd:element name="identifier" type="xsd:anyURI"
5137 maxOccurs="unbounded"/>
5138             </xsd:sequence>
5139         </xsd:complexType>
5140     </xsd:element>
5141     <xsd:element name="batchClaimResponse">
5142         <xsd:complexType>
5143             <xsd:sequence>
5144                 <xsd:element name="batchResponse" type="tBatchResponse"
5145 minOccurs="0" maxOccurs="unbounded"/>
5146             </xsd:sequence>
5147         </xsd:complexType>
5148     </xsd:element>
5149
5150     <xsd:element name="complete">
5151         <xsd:complexType>
5152             <xsd:sequence>
5153                 <xsd:element name="identifier" type="xsd:anyURI"/>
5154                 <xsd:element name="taskData" type="xsd:anyType" minOccurs="0"/>
5155             </xsd:sequence>
5156         </xsd:complexType>
5157     </xsd:element>

```

```

5158     <xsd:element name="completeResponse">
5159         <xsd:complexType>
5160             <xsd:sequence>
5161                 <xsd:annotation>
5162                     <xsd:documentation>Empty message</xsd:documentation>
5163                 </xsd:annotation>
5164             </xsd:sequence>
5165         </xsd:complexType>
5166     </xsd:element>
5167
5168     <xsd:element name="batchComplete">
5169         <xsd:complexType>
5170             <xsd:sequence>
5171                 <xsd:element name="identifier" type="xsd:anyURI"
5172                 minOccurs="unbounded" />
5173
5174                 <xsd:element name="taskData" type="xsd:anyType" minOccurs="0" />
5175             </xsd:sequence>
5176         </xsd:complexType>
5177     </xsd:element>
5178     <xsd:element name="batchCompleteResponse">
5179         <xsd:complexType>
5180             <xsd:sequence>
5181                 <xsd:element name="batchResponse" type="tBatchResponse"
5182                 minOccurs="0" maxOccurs="unbounded" />
5183             </xsd:sequence>
5184         </xsd:complexType>
5185     </xsd:element>
5186
5187     <xsd:element name="delegate">
5188         <xsd:complexType>
5189             <xsd:sequence>
5190                 <xsd:element name="identifier" type="xsd:anyURI" />
5191                 <xsd:element name="organizationalEntity"
5192                 type="htt:tOrganizationalEntity" />
5193             </xsd:sequence>
5194         </xsd:complexType>
5195     </xsd:element>
5196     <xsd:element name="delegateResponse">
5197         <xsd:complexType>
5198             <xsd:sequence>
5199                 <xsd:annotation>
5200                     <xsd:documentation>Empty message</xsd:documentation>
5201                 </xsd:annotation>
5202             </xsd:sequence>
5203         </xsd:complexType>
5204     </xsd:element>
5205
5206     <xsd:element name="batchDelegate">
5207         <xsd:complexType>
5208             <xsd:sequence>
5209                 <xsd:element name="identifier" type="xsd:anyURI"
5210                 maxOccurs="unbounded" />
5211                 <xsd:element name="organizationalEntity"
5212                 type="htt:tOrganizationalEntity" />
5213             </xsd:sequence>
5214         </xsd:complexType>
5215     </xsd:element>

```

```

5216     <xsd:element name="batchDelegateResponse">
5217         <xsd:complexType>
5218             <xsd:sequence>
5219                 <xsd:element name="batchResponse" type="tBatchResponse"
5220 minOccurs="0" maxOccurs="unbounded"/>
5221             </xsd:sequence>
5222         </xsd:complexType>
5223     </xsd:element>
5224
5225     <xsd:element name="deleteAttachment">
5226         <xsd:complexType>
5227             <xsd:sequence>
5228                 <xsd:element name="taskIdentifier" type="xsd:anyURI"/>
5229                 <xsd:element name="attachmentIdentifier" type="xsd:anyURI"/>
5230             </xsd:sequence>
5231         </xsd:complexType>
5232     </xsd:element>
5233     <xsd:element name="deleteAttachmentResponse">
5234         <xsd:complexType>
5235             <xsd:sequence>
5236                 <xsd:annotation>
5237                     <xsd:documentation>Empty message</xsd:documentation>
5238                 </xsd:annotation>
5239             </xsd:sequence>
5240         </xsd:complexType>
5241     </xsd:element>
5242
5243     <xsd:element name="deleteComment">
5244         <xsd:complexType>
5245             <xsd:sequence>
5246                 <xsd:element name="taskIdentifier" type="xsd:anyURI"/>
5247                 <xsd:element name="commentIdentifier" type="xsd:anyURI"/>
5248             </xsd:sequence>
5249         </xsd:complexType>
5250     </xsd:element>
5251     <xsd:element name="deleteCommentResponse">
5252         <xsd:complexType>
5253             <xsd:sequence>
5254                 <xsd:annotation>
5255                     <xsd:documentation>Empty message</xsd:documentation>
5256                 </xsd:annotation>
5257             </xsd:sequence>
5258         </xsd:complexType>
5259     </xsd:element>
5260
5261     <xsd:element name="deleteFault">
5262         <xsd:complexType>
5263             <xsd:sequence>
5264                 <xsd:element name="identifier" type="xsd:anyURI"/>
5265             </xsd:sequence>
5266         </xsd:complexType>
5267     </xsd:element>
5268     <xsd:element name="deleteFaultResponse">
5269         <xsd:complexType>
5270             <xsd:sequence>
5271                 <xsd:annotation>
5272                     <xsd:documentation>Empty message</xsd:documentation>
5273                 </xsd:annotation>

```

```

5274     </xsd:sequence>
5275 </xsd:complexType>
5276 </xsd:element>
5277
5278 <xsd:element name="deleteOutput">
5279 <xsd:complexType>
5280 <xsd:sequence>
5281 <xsd:element name="identifier" type="xsd:anyURI" />
5282 </xsd:sequence>
5283 </xsd:complexType>
5284 </xsd:element>
5285 <xsd:element name="deleteOutputResponse">
5286 <xsd:complexType>
5287 <xsd:sequence>
5288 <xsd:annotation>
5289 <xsd:documentation>Empty message</xsd:documentation>
5290 </xsd:annotation>
5291 </xsd:sequence>
5292 </xsd:complexType>
5293 </xsd:element>
5294
5295 <xsd:element name="fail">
5296 <xsd:complexType>
5297 <xsd:sequence>
5298 <xsd:element name="identifier" type="xsd:anyURI" />
5299 <xsd:element name="fault" type="htt:tFault" minOccurs="0" />
5300 </xsd:sequence>
5301 </xsd:complexType>
5302 </xsd:element>
5303 <xsd:element name="failResponse">
5304 <xsd:complexType>
5305 <xsd:sequence>
5306 <xsd:annotation>
5307 <xsd:documentation>Empty message</xsd:documentation>
5308 </xsd:annotation>
5309 </xsd:sequence>
5310 </xsd:complexType>
5311 </xsd:element>
5312
5313 <xsd:element name="batchFail">
5314 <xsd:complexType>
5315 <xsd:sequence>
5316 <xsd:element name="identifier" type="xsd:anyURI"
5317 minOccurs="unbounded" />
5318 <xsd:element name="fault" type="htt:tFault" minOccurs="0" />
5319 </xsd:sequence>
5320 </xsd:complexType>
5321 </xsd:element>
5322 <xsd:element name="batchFailResponse">
5323 <xsd:complexType>
5324 <xsd:sequence>
5325 <xsd:element name="batchResponse" type="tBatchResponse"
5326 minOccurs="0" maxOccurs="unbounded" />
5327 </xsd:sequence>
5328 </xsd:complexType>
5329 </xsd:element>
5330
5331 <xsd:element name="forward">

```

```

5332     <xsd:complexType>
5333     <xsd:sequence>
5334     <xsd:element name="identifier" type="xsd:anyURI" />
5335     <xsd:element name="organizationalEntity"
5336     type="htt:tOrganizationalEntity" />
5337     </xsd:sequence>
5338     </xsd:complexType>
5339     </xsd:element>
5340     <xsd:element name="forwardResponse">
5341     <xsd:complexType>
5342     <xsd:sequence>
5343     <xsd:annotation>
5344     <xsd:documentation>Empty message</xsd:documentation>
5345     </xsd:annotation>
5346     </xsd:sequence>
5347     </xsd:complexType>
5348     </xsd:element>
5349
5350     <xsd:element name="batchForward">
5351     <xsd:complexType>
5352     <xsd:sequence>
5353     <xsd:element name="identifier" type="xsd:anyURI"
5354     minOccurs="unbounded" />
5355     <xsd:element name="organizationalEntity"
5356     type="htt:tOrganizationalEntity" />
5357     </xsd:sequence>
5358     </xsd:complexType>
5359     </xsd:element>
5360     <xsd:element name="batchForwardResponse">
5361     <xsd:complexType>
5362     <xsd:sequence>
5363     <xsd:element name="batchResponse" type="tBatchResponse"
5364     minOccurs="0" maxOccurs="unbounded" />
5365     </xsd:sequence>
5366     </xsd:complexType>
5367     </xsd:element>
5368
5369     <xsd:element name="getAttachment">
5370     <xsd:complexType>
5371     <xsd:sequence>
5372     <xsd:element name="taskIdentifier" type="xsd:anyURI" />
5373     <xsd:element name="attachmentIdentifier" type="xsd:anyURI" />
5374     </xsd:sequence>
5375     </xsd:complexType>
5376     </xsd:element>
5377     <xsd:element name="getAttachmentResponse">
5378     <xsd:complexType>
5379     <xsd:sequence>
5380     <xsd:element name="attachment" type="htt:tAttachment"
5381     minOccurs="0" maxOccurs="unbounded" />
5382     </xsd:sequence>
5383     </xsd:complexType>
5384     </xsd:element>
5385
5386     <xsd:element name="getAttachmentInfos">
5387     <xsd:complexType>
5388     <xsd:sequence>
5389     <xsd:element name="identifier" type="xsd:anyURI" />

```

```

5390         </xsd:sequence>
5391     </xsd:complexType>
5392 </xsd:element>
5393 <xsd:element name="getAttachmentInfosResponse">
5394     <xsd:complexType>
5395         <xsd:sequence>
5396             <xsd:element name="info" type="htt:tAttachmentInfo" minOccurs="0"
5397 maxOccurs="unbounded" />
5398         </xsd:sequence>
5399     </xsd:complexType>
5400 </xsd:element>
5401
5402 <xsd:element name="getComments">
5403     <xsd:complexType>
5404         <xsd:sequence>
5405             <xsd:element name="identifier" type="xsd:anyURI" />
5406         </xsd:sequence>
5407     </xsd:complexType>
5408 </xsd:element>
5409 <xsd:element name="getCommentsResponse">
5410     <xsd:complexType>
5411         <xsd:sequence>
5412             <xsd:element name="comment" type="htt:tComment" minOccurs="0"
5413 maxOccurs="unbounded" />
5414         </xsd:sequence>
5415     </xsd:complexType>
5416 </xsd:element>
5417
5418 <xsd:element name="getFault">
5419     <xsd:complexType>
5420         <xsd:sequence>
5421             <xsd:element name="identifier" type="xsd:anyURI" />
5422         </xsd:sequence>
5423     </xsd:complexType>
5424 </xsd:element>
5425 <xsd:element name="getFaultResponse">
5426     <xsd:complexType>
5427         <xsd:sequence>
5428             <xsd:element name="fault" type="htt:tFault" />
5429         </xsd:sequence>
5430     </xsd:complexType>
5431 </xsd:element>
5432
5433 <xsd:element name="getInput">
5434     <xsd:complexType>
5435         <xsd:sequence>
5436             <xsd:element name="identifier" type="xsd:anyURI" />
5437             <xsd:element name="part" type="xsd:NCName" minOccurs="0" />
5438         </xsd:sequence>
5439     </xsd:complexType>
5440 </xsd:element>
5441 <xsd:element name="getInputResponse">
5442     <xsd:complexType>
5443         <xsd:sequence>
5444             <xsd:element name="taskData" type="xsd:anyType" />
5445         </xsd:sequence>
5446     </xsd:complexType>
5447 </xsd:element>

```



```

5448 <xsd:element name="getOutcome">
5449 <xsd:complexType>
5450 <xsd:sequence>
5451 <xsd:element name="identifier" type="xsd:anyURI"/>
5452 </xsd:sequence>
5453 </xsd:complexType>
5454 </xsd:element>
5455 <xsd:element name="getOutcomeResponse">
5456 <xsd:complexType>
5457 <xsd:sequence>
5458 <xsd:element name="outcome" type="xsd:string"/>
5459 </xsd:sequence>
5460 </xsd:complexType>
5461 </xsd:element>
5462
5463 <xsd:element name="getOutput">
5464 <xsd:complexType>
5465 <xsd:sequence>
5466 <xsd:element name="identifier" type="xsd:anyURI"/>
5467 <xsd:element name="part" type="xsd:NCName" minOccurs="0"/>
5468 </xsd:sequence>
5469 </xsd:complexType>
5470 </xsd:element>
5471 <xsd:element name="getOutputResponse">
5472 <xsd:complexType>
5473 <xsd:sequence>
5474 <xsd:element name="taskData" type="xsd:anyType"/>
5475 </xsd:sequence>
5476 </xsd:complexType>
5477 </xsd:element>
5478
5479 <xsd:element name="getParentTask">
5480 <xsd:complexType>
5481 <xsd:sequence>
5482 <xsd:element name="taskIdentifier" type="xsd:anyURI"/>
5483 </xsd:sequence>
5484 </xsd:complexType>
5485 </xsd:element>
5486 <xsd:element name="getParentTaskResponse">
5487 <xsd:complexType>
5488 <xsd:sequence>
5489 <xsd:element name="parentTask" type="htt:tTask" minOccurs="0"/>
5490 </xsd:sequence>
5491 </xsd:complexType>
5492 </xsd:element>
5493
5494 <xsd:element name="getParentTaskIdentifier">
5495 <xsd:complexType>
5496 <xsd:sequence>
5497 <xsd:element name="taskIdentifier" type="xsd:anyURI"/>
5498 </xsd:sequence>
5499 </xsd:complexType>
5500 </xsd:element>
5501 <xsd:element name="getParentTaskIdentifierResponse">
5502 <xsd:complexType>
5503 <xsd:sequence>
5504

```

```

5505     <xsd:element name="parentTaskIdentifier" type="xsd:anyURI"
5506 minOccurs="0" />
5507     </xsd:sequence>
5508   </xsd:complexType>
5509 </xsd:element>
5510
5511   <xsd:element name="getRendering">
5512     <xsd:complexType>
5513       <xsd:sequence>
5514         <xsd:element name="identifier" type="xsd:anyURI" />
5515         <xsd:element name="renderingType" type="xsd:QName" />
5516       </xsd:sequence>
5517     </xsd:complexType>
5518   </xsd:element>
5519   <xsd:element name="getRenderingResponse">
5520     <xsd:complexType>
5521       <xsd:sequence>
5522         <xsd:element name="rendering" type="xsd:anyType" />
5523       </xsd:sequence>
5524     </xsd:complexType>
5525   </xsd:element>
5526
5527   <xsd:element name="getRenderingTypes">
5528     <xsd:complexType>
5529       <xsd:sequence>
5530         <xsd:element name="identifier" type="xsd:anyURI" />
5531       </xsd:sequence>
5532     </xsd:complexType>
5533   </xsd:element>
5534   <xsd:element name="getRenderingTypesResponse">
5535     <xsd:complexType>
5536       <xsd:sequence>
5537         <xsd:element name="renderingType" type="xsd:QName" minOccurs="0"
5538 maxOccurs="unbounded" />
5539       </xsd:sequence>
5540     </xsd:complexType>
5541   </xsd:element>
5542
5543   <xsd:element name="getSubtaskIdentifiers">
5544     <xsd:complexType>
5545       <xsd:sequence>
5546         <xsd:element name="taskIdentifier" type="xsd:anyURI" />
5547       </xsd:sequence>
5548     </xsd:complexType>
5549   </xsd:element>
5550   <xsd:element name="getSubtaskIdentifiersResponse">
5551     <xsd:complexType>
5552       <xsd:sequence>
5553         <xsd:element name="subtaskIdentifier" type="xsd:anyURI"
5554 minOccurs="0" maxOccurs="unbounded" />
5555       </xsd:sequence>
5556     </xsd:complexType>
5557   </xsd:element>
5558
5559   <xsd:element name="getSubtasks">
5560     <xsd:complexType>
5561       <xsd:sequence>
5562         <xsd:element name="taskIdentifier" type="xsd:anyURI" />

```

```

5563     </xsd:sequence>
5564   </xsd:complexType>
5565 </xsd:element>
5566 <xsd:element name="getSubtasksResponse">
5567   <xsd:complexType>
5568     <xsd:sequence>
5569       <xsd:element name="subtask" type="htt:tTask" minOccurs="0"
5570 maxOccurs="unbounded" />
5571     </xsd:sequence>
5572   </xsd:complexType>
5573 </xsd:element>
5574
5575 <xsd:element name="getTaskDescription">
5576   <xsd:complexType>
5577     <xsd:sequence>
5578       <xsd:element name="identifier" type="xsd:anyURI" />
5579       <xsd:element name="contentType" type="xsd:string" minOccurs="0" />
5580     </xsd:sequence>
5581   </xsd:complexType>
5582 </xsd:element>
5583 <xsd:element name="getTaskDescriptionResponse">
5584   <xsd:complexType>
5585     <xsd:sequence>
5586       <xsd:element name="description" type="xsd:string" />
5587     </xsd:sequence>
5588   </xsd:complexType>
5589 </xsd:element>
5590
5591 <xsd:element name="getTaskDetails">
5592   <xsd:complexType>
5593     <xsd:sequence>
5594       <xsd:element name="identifier" type="xsd:anyURI" />
5595     </xsd:sequence>
5596   </xsd:complexType>
5597 </xsd:element>
5598 <xsd:element name="getTaskDetailsResponse">
5599   <xsd:complexType>
5600     <xsd:sequence>
5601       <xsd:element name="taskDetails" type="htt:tTaskDetails" />
5602     </xsd:sequence>
5603   </xsd:complexType>
5604 </xsd:element>
5605
5606 <xsd:element name="getTaskHistory">
5607   <xsd:complexType>
5608     <xsd:sequence>
5609       <xsd:element name="identifier" type="xsd:anyURI" />
5610       <xsd:element name="filter" type="htt:tTaskHistoryFilter"
5611 minOccurs="0" />
5612       <xsd:element name="startIndex" type="xsd:int" minOccurs="0" />
5613       <xsd:element name="maxTasks" type="xsd:int" minOccurs="0" />
5614     </xsd:sequence>
5615     <xsd:attribute name="includeData" type="xsd:boolean" />
5616   </xsd:complexType>
5617 </xsd:element>
5618 <xsd:element name="getTaskHistoryResponse">
5619   <xsd:complexType>
5620     <xsd:sequence>

```

```

5621         <xsd:element name="taskEvent" type="htt:tTaskEventType"
5622 minOccurs="0" maxOccurs="unbounded" />
5623     </xsd:sequence>
5624 </xsd:complexType>
5625 </xsd:element>
5626
5627     <xsd:element name="getTaskInstanceData">
5628     <xsd:complexType>
5629     <xsd:sequence>
5630     <xsd:element name="identifier" type="xsd:anyURI" />
5631     <xsd:element name="properties" type="xsd:string" />
5632     <xsd:element name="renderingPreferences"
5633 type="htt:tRenderingTypes" minOccurs="0" maxOccurs="unbounded" />
5634     </xsd:sequence>
5635     </xsd:complexType>
5636     </xsd:element>
5637     <xsd:element name="getTaskInstanceDataResponse">
5638     <xsd:complexType>
5639     <xsd:sequence>
5640     <xsd:element name="taskInstanceData"
5641 type="htt:tTaskInstanceData" />
5642     </xsd:sequence>
5643     </xsd:complexType>
5644     </xsd:element>
5645
5646     <xsd:element name="getTaskOperations">
5647     <xsd:complexType>
5648     <xsd:sequence>
5649     <xsd:element name="identifier" type="xsd:anyURI" />
5650     </xsd:sequence>
5651     </xsd:complexType>
5652     </xsd:element>
5653     <xsd:element name="getTaskOperationsResponse">
5654     <xsd:complexType>
5655     <xsd:sequence>
5656     <xsd:element name="taskOperations" type="htt:tTaskOperations" />
5657     </xsd:sequence>
5658     </xsd:complexType>
5659     </xsd:element>
5660
5661     <xsd:element name="hasSubtasks">
5662     <xsd:complexType>
5663     <xsd:sequence>
5664     <xsd:element name="taskIdentifier" type="xsd:anyURI" />
5665     </xsd:sequence>
5666     </xsd:complexType>
5667     </xsd:element>
5668     <xsd:element name="hasSubtasksResponse">
5669     <xsd:complexType>
5670     <xsd:sequence>
5671     <xsd:element name="result" type="xsd:boolean" />
5672     </xsd:sequence>
5673     </xsd:complexType>
5674     </xsd:element>
5675
5676     <xsd:element name="instantiateSubtask">
5677     <xsd:complexType>
5678     <xsd:sequence>

```

```

5679     <xsd:element name="taskIdentifier" type="xsd:anyURI" />
5680     <xsd:element name="name" type="xsd:string" />
5681   </xsd:sequence>
5682 </xsd:complexType>
5683 </xsd:element>
5684 <xsd:element name="instantiateSubtaskResponse">
5685   <xsd:complexType>
5686     <xsd:sequence>
5687       <xsd:element name="subtaskIdentifier" type="xsd:anyURI" />
5688     </xsd:sequence>
5689   </xsd:complexType>
5690 </xsd:element>
5691
5692 <xsd:element name="isSubtask">
5693   <xsd:complexType>
5694     <xsd:sequence>
5695       <xsd:element name="taskIdentifier" type="xsd:anyURI" />
5696     </xsd:sequence>
5697   </xsd:complexType>
5698 </xsd:element>
5699 <xsd:element name="isSubtaskResponse">
5700   <xsd:complexType>
5701     <xsd:sequence>
5702       <xsd:element name="result" type="xsd:boolean" />
5703     </xsd:sequence>
5704   </xsd:complexType>
5705 </xsd:element>
5706
5707 <xsd:element name="release">
5708   <xsd:complexType>
5709     <xsd:sequence>
5710       <xsd:element name="identifier" type="xsd:anyURI" />
5711     </xsd:sequence>
5712   </xsd:complexType>
5713 </xsd:element>
5714 <xsd:element name="releaseResponse">
5715   <xsd:complexType>
5716     <xsd:sequence>
5717       <xsd:annotation>
5718         <xsd:documentation>Empty message</xsd:documentation>
5719       </xsd:annotation>
5720     </xsd:sequence>
5721   </xsd:complexType>
5722 </xsd:element>
5723
5724 <xsd:element name="batchRelease">
5725   <xsd:complexType>
5726     <xsd:sequence>
5727       <xsd:element name="identifier" type="xsd:anyURI"
5728 maxOccurs="unbounded" />
5729     </xsd:sequence>
5730   </xsd:complexType>
5731 </xsd:element>
5732 <xsd:element name="batchReleaseResponse">
5733   <xsd:complexType>
5734     <xsd:sequence>
5735       <xsd:element name="batchResponse" type="tBatchResponse"
5736 minOccurs="0" maxOccurs="unbounded" />

```

```

5737     </xsd:sequence>
5738 </xsd:complexType>
5739 </xsd:element>
5740
5741 <xsd:element name="remove">
5742 <xsd:complexType>
5743 <xsd:sequence>
5744 <xsd:element name="identifier" type="xsd:anyURI" />
5745 </xsd:sequence>
5746 </xsd:complexType>
5747 </xsd:element>
5748 <xsd:element name="removeResponse">
5749 <xsd:complexType>
5750 <xsd:sequence>
5751 <xsd:annotation>
5752 <xsd:documentation>Empty message</xsd:documentation>
5753 </xsd:annotation>
5754 </xsd:sequence>
5755 </xsd:complexType>
5756 </xsd:element>
5757
5758 <xsd:element name="batchRemove">
5759 <xsd:complexType>
5760 <xsd:sequence>
5761 <xsd:element name="identifier" type="xsd:anyURI"
5762 maxOccurs="unbounded" />
5763 </xsd:sequence>
5764 </xsd:complexType>
5765 </xsd:element>
5766 <xsd:element name="batchRemoveResponse">
5767 <xsd:complexType>
5768 <xsd:sequence>
5769 <xsd:element name="batchResponse" type="tBatchResponse"
5770 minOccurs="0" maxOccurs="unbounded" />
5771 </xsd:sequence>
5772 </xsd:complexType>
5773 </xsd:element>
5774
5775 <xsd:element name="resume">
5776 <xsd:complexType>
5777 <xsd:sequence>
5778 <xsd:element name="identifier" type="xsd:anyURI" />
5779 </xsd:sequence>
5780 </xsd:complexType>
5781 </xsd:element>
5782 <xsd:element name="resumeResponse">
5783 <xsd:complexType>
5784 <xsd:sequence>
5785 <xsd:annotation>
5786 <xsd:documentation>Empty message</xsd:documentation>
5787 </xsd:annotation>
5788 </xsd:sequence>
5789 </xsd:complexType>
5790 </xsd:element>
5791
5792 <xsd:element name="batchResume">
5793 <xsd:complexType>
5794 <xsd:sequence>

```

```

5795         <xsd:element name="identifier" type="xsd:anyURI"
5796 maxOccurs="unbounded" />
5797     </xsd:sequence>
5798 </xsd:complexType>
5799 </xsd:element>
5800 <xsd:element name="batchResumeResponse">
5801 <xsd:complexType>
5802 <xsd:sequence>
5803 <xsd:element name="batchResponse" type="tBatchResponse"
5804 minOccurs="0" maxOccurs="unbounded" />
5805 </xsd:sequence>
5806 </xsd:complexType>
5807 </xsd:element>
5808
5809 <xsd:element name="setFault">
5810 <xsd:complexType>
5811 <xsd:sequence>
5812 <xsd:element name="identifier" type="xsd:anyURI" />
5813 <xsd:element name="fault" type="htt:tFault" />
5814 </xsd:sequence>
5815 </xsd:complexType>
5816 </xsd:element>
5817 <xsd:element name="setFaultResponse">
5818 <xsd:complexType>
5819 <xsd:sequence>
5820 <xsd:annotation>
5821 <xsd:documentation>Empty message</xsd:documentation>
5822 </xsd:annotation>
5823 </xsd:sequence>
5824 </xsd:complexType>
5825 </xsd:element>
5826
5827 <xsd:element name="setOutput">
5828 <xsd:complexType>
5829 <xsd:sequence>
5830 <xsd:element name="identifier" type="xsd:anyURI" />
5831 <xsd:element name="part" type="xsd:NCName" minOccurs="0" />
5832 <xsd:element name="taskData" type="xsd:anyType" />
5833 </xsd:sequence>
5834 </xsd:complexType>
5835 </xsd:element>
5836 <xsd:element name="setOutputResponse">
5837 <xsd:complexType>
5838 <xsd:sequence>
5839 <xsd:annotation>
5840 <xsd:documentation>Empty message</xsd:documentation>
5841 </xsd:annotation>
5842 </xsd:sequence>
5843 </xsd:complexType>
5844 </xsd:element>
5845
5846 <xsd:element name="setPriority">
5847 <xsd:complexType>
5848 <xsd:sequence>
5849 <xsd:element name="identifier" type="xsd:anyURI" />
5850 <xsd:element name="priority" type="htt:tPriority" />
5851 </xsd:sequence>
5852 </xsd:complexType>

```

```

5853     </xsd:element>
5854     <xsd:element name="setPriorityResponse">
5855         <xsd:complexType>
5856             <xsd:sequence>
5857                 <xsd:annotation>
5858                     <xsd:documentation>Empty message</xsd:documentation>
5859                 </xsd:annotation>
5860             </xsd:sequence>
5861         </xsd:complexType>
5862     </xsd:element>
5863
5864     <xsd:element name="batchSetPriority">
5865         <xsd:complexType>
5866             <xsd:sequence>
5867                 <xsd:element name="identifier" type="xsd:anyURI"
5868 minOccurs="unbounded" />
5869                 <xsd:element name="priority" type="htt:tPriority" />
5870             </xsd:sequence>
5871         </xsd:complexType>
5872     </xsd:element>
5873     <xsd:element name="batchSetPriorityResponse">
5874         <xsd:complexType>
5875             <xsd:sequence>
5876                 <xsd:element name="batchResponse" type="tBatchResponse"
5877 minOccurs="0" maxOccurs="unbounded" />
5878             </xsd:sequence>
5879         </xsd:complexType>
5880     </xsd:element>
5881
5882     <xsd:element name="setTaskCompletionDeadlineExpression">
5883         <xsd:complexType>
5884             <xsd:sequence>
5885                 <xsd:element name="identifier" type="xsd:anyURI" />
5886                 <xsd:element name="deadlineName" type="xsd:NCName" />
5887                 <xsd:element name="deadlineExpression" type="xsd:string" />
5888             </xsd:sequence>
5889         </xsd:complexType>
5890     </xsd:element>
5891     <xsd:element name="setTaskCompletionDeadlineExpressionResponse">
5892         <xsd:complexType>
5893             <xsd:sequence>
5894                 <xsd:annotation>
5895                     <xsd:documentation>Empty message</xsd:documentation>
5896                 </xsd:annotation>
5897             </xsd:sequence>
5898         </xsd:complexType>
5899     </xsd:element>
5900
5901     <xsd:element name="setTaskCompletionDurationExpression">
5902         <xsd:complexType>
5903             <xsd:sequence>
5904                 <xsd:element name="identifier" type="xsd:anyURI" />
5905                 <xsd:element name="deadlineName" type="xsd:NCName" />
5906                 <xsd:element name="durationExpression" type="xsd:string" />
5907             </xsd:sequence>
5908         </xsd:complexType>
5909     </xsd:element>
5910     <xsd:element name="setTaskCompletionDurationExpressionResponse">

```



```

5911     <xsd:complexType>
5912     <xsd:sequence>
5913     <xsd:annotation>
5914     <xsd:documentation>Empty message</xsd:documentation>
5915     </xsd:annotation>
5916     </xsd:sequence>
5917     </xsd:complexType>
5918 </xsd:element>
5919
5920 <xsd:element name="setTaskStartDeadlineExpression">
5921 <xsd:complexType>
5922 <xsd:sequence>
5923 <xsd:element name="identifier" type="xsd:anyURI"/>
5924 <xsd:element name="deadlineName" type="xsd:NCName"/>
5925 <xsd:element name="deadlineExpression" type="xsd:string"/>
5926 </xsd:sequence>
5927 </xsd:complexType>
5928 </xsd:element>
5929 <xsd:element name="setTaskStartDeadlineExpressionResponse">
5930 <xsd:complexType>
5931 <xsd:sequence>
5932 <xsd:annotation>
5933 <xsd:documentation>Empty message</xsd:documentation>
5934 </xsd:annotation>
5935 </xsd:sequence>
5936 </xsd:complexType>
5937 </xsd:element>
5938
5939 <xsd:element name="setTaskStartDurationExpression">
5940 <xsd:complexType>
5941 <xsd:sequence>
5942 <xsd:element name="identifier" type="xsd:anyURI"/>
5943 <xsd:element name="deadlineName" type="xsd:NCName"/>
5944 <xsd:element name="durationExpression" type="xsd:string"/>
5945 </xsd:sequence>
5946 </xsd:complexType>
5947 </xsd:element>
5948 <xsd:element name="setTaskStartDurationExpressionResponse">
5949 <xsd:complexType>
5950 <xsd:sequence>
5951 <xsd:annotation>
5952 <xsd:documentation>Empty message</xsd:documentation>
5953 </xsd:annotation>
5954 </xsd:sequence>
5955 </xsd:complexType>
5956 </xsd:element>
5957
5958 <xsd:element name="skip">
5959 <xsd:complexType>
5960 <xsd:sequence>
5961 <xsd:element name="identifier" type="xsd:anyURI"/>
5962 </xsd:sequence>
5963 </xsd:complexType>
5964 </xsd:element>
5965 <xsd:element name="skipResponse">
5966 <xsd:complexType>
5967 <xsd:sequence>
5968 <xsd:annotation>

```

```

5969         <xsd:documentation>Empty message</xsd:documentation>
5970     </xsd:annotation>
5971 </xsd:sequence>
5972 </xsd:complexType>
5973 </xsd:element>
5974
5975     <xsd:element name="batchSkip">
5976         <xsd:complexType>
5977             <xsd:sequence>
5978                 <xsd:element name="identifier" type="xsd:anyURI"
5979 maxOccurs="unbounded" />
5980             </xsd:sequence>
5981         </xsd:complexType>
5982     </xsd:element>
5983     <xsd:element name="batchSkipResponse">
5984         <xsd:complexType>
5985             <xsd:sequence>
5986                 <xsd:element name="batchResponse" type="tBatchResponse"
5987 minOccurs="0" maxOccurs="unbounded" />
5988             </xsd:sequence>
5989         </xsd:complexType>
5990     </xsd:element>
5991
5992     <xsd:element name="start">
5993         <xsd:complexType>
5994             <xsd:sequence>
5995                 <xsd:element name="identifier" type="xsd:anyURI" />
5996             </xsd:sequence>
5997         </xsd:complexType>
5998     </xsd:element>
5999     <xsd:element name="startResponse">
6000         <xsd:complexType>
6001             <xsd:sequence>
6002                 <xsd:annotation>
6003                     <xsd:documentation>Empty message</xsd:documentation>
6004                 </xsd:annotation>
6005             </xsd:sequence>
6006         </xsd:complexType>
6007     </xsd:element>
6008
6009     <xsd:element name="batchStart">
6010         <xsd:complexType>
6011             <xsd:sequence>
6012                 <xsd:element name="identifier" type="xsd:anyURI"
6013 maxOccurs="unbounded" />
6014             </xsd:sequence>
6015         </xsd:complexType>
6016     </xsd:element>
6017     <xsd:element name="batchStartResponse">
6018         <xsd:complexType>
6019             <xsd:sequence>
6020                 <xsd:element name="batchResponse" type="tBatchResponse"
6021 minOccurs="0" maxOccurs="unbounded" />
6022             </xsd:sequence>
6023         </xsd:complexType>
6024     </xsd:element>
6025
6026     <xsd:element name="stop">

```

```

6027     <xsd:complexType>
6028         <xsd:sequence>
6029             <xsd:element name="identifier" type="xsd:anyURI" />
6030         </xsd:sequence>
6031     </xsd:complexType>
6032 </xsd:element>
6033 <xsd:element name="stopResponse">
6034     <xsd:complexType>
6035         <xsd:sequence>
6036             <xsd:annotation>
6037                 <xsd:documentation>Empty message</xsd:documentation>
6038             </xsd:annotation>
6039         </xsd:sequence>
6040     </xsd:complexType>
6041 </xsd:element>
6042
6043     <xsd:element name="batchStop">
6044         <xsd:complexType>
6045             <xsd:sequence>
6046                 <xsd:element name="identifier" type="xsd:anyURI"
6047 maxOccurs="unbounded" />
6048             </xsd:sequence>
6049         </xsd:complexType>
6050     </xsd:element>
6051     <xsd:element name="batchStopResponse">
6052         <xsd:complexType>
6053             <xsd:sequence>
6054                 <xsd:element name="batchResponse" type="tBatchResponse"
6055 minOccurs="0" maxOccurs="unbounded" />
6056             </xsd:sequence>
6057         </xsd:complexType>
6058     </xsd:element>
6059
6060 <xsd:element name="release">
6061 <xsd:complexType>
6062 <xsd:sequence>
6063 <xsd:element name="identifier" type="xsd:anyURI" />
6064 </xsd:sequence>
6065 </xsd:complexType>
6066 </xsd:element>
6067 <xsd:element name="releaseResponse">
6068 <xsd:complexType>
6069 <xsd:sequence>
6070 <xsd:annotation>
6071 <xsd:documentation>Empty message</xsd:documentation>
6072 </xsd:annotation>
6073 </xsd:sequence>
6074 </xsd:complexType>
6075 </xsd:element>
6076
6077 <xsd:element name="batchRelease">
6078 <xsd:complexType>
6079 <xsd:sequence>
6080 <xsd:element name="identifier" type="xsd:anyURI"
6081 maxOccurs="unbounded" />
6082 </xsd:sequence>
6083 </xsd:complexType>
6084 </xsd:element>

```

```

6085 <xsd:element name="batchReleaseResponse">
6086 <xsd:complexType>
6087 <xsd:sequence>
6088 <xsd:element name="batchResponse" type="tBatchResponse"
6089 minOccurs="0" maxOccurs="unbounded"/>
6090 </xsd:sequence>
6091 </xsd:complexType>
6092 </xsd:element>
6093
6094 <xsd:element name="suspend">
6095   <xsd:complexType>
6096     <xsd:sequence>
6097       <xsd:element name="identifier" type="xsd:anyURI"/>
6098     </xsd:sequence>
6099   </xsd:complexType>
6100 </xsd:element>
6101 <xsd:element name="suspendResponse">
6102   <xsd:complexType>
6103     <xsd:sequence>
6104       <xsd:annotation>
6105         <xsd:documentation>Empty message</xsd:documentation>
6106       </xsd:annotation>
6107     </xsd:sequence>
6108   </xsd:complexType>
6109 </xsd:element>
6110
6111 <xsd:element name="batchSuspend">
6112   <xsd:complexType>
6113     <xsd:sequence>
6114       <xsd:element name="identifier" type="xsd:anyURI"
6115 maxOccurs="unbounded"/>
6116     </xsd:sequence>
6117   </xsd:complexType>
6118 </xsd:element>
6119 <xsd:element name="batchSuspendResponse">
6120   <xsd:complexType>
6121     <xsd:sequence>
6122       <xsd:element name="batchResponse" type="tBatchResponse"
6123 minOccurs="0" maxOccurs="unbounded"/>
6124     </xsd:sequence>
6125   </xsd:complexType>
6126 </xsd:element>
6127
6128 <xsd:element name="suspendUntil">
6129   <xsd:complexType>
6130     <xsd:sequence>
6131       <xsd:element name="identifier" type="xsd:anyURI"/>
6132       <xsd:element name="time" type="htt:tTime"/>
6133     </xsd:sequence>
6134   </xsd:complexType>
6135 </xsd:element>
6136 <xsd:element name="suspendUntilResponse">
6137   <xsd:complexType>
6138     <xsd:sequence>
6139       <xsd:annotation>
6140         <xsd:documentation>Empty message</xsd:documentation>
6141       </xsd:annotation>
6142     </xsd:sequence>

```

```

6143     </xsd:complexType>
6144 </xsd:element>
6145
6146     <xsd:element name="batchSuspendUntil">
6147         <xsd:complexType>
6148             <xsd:sequence>
6149                 <xsd:element name="identifier" type="xsd:anyURI"
6150 maxOccurs="unbounded" />
6151                 <xsd:element name="time" type="htt:tTime" />
6152             </xsd:sequence>
6153         </xsd:complexType>
6154     </xsd:element>
6155     <xsd:element name="batchSuspendUntilResponse">
6156         <xsd:complexType>
6157             <xsd:sequence>
6158                 <xsd:element name="batchResponse" type="tBatchResponse"
6159 minOccurs="0" maxOccurs="unbounded" />
6160             </xsd:sequence>
6161         </xsd:complexType>
6162     </xsd:element>
6163
6164     <xsd:element name="resume">
6165     <xsd:complexType>
6166     <xsd:sequence>
6167     <xsd:element name="identifier" type="xsd:anyURI" />
6168     </xsd:sequence>
6169     </xsd:complexType>
6170     </xsd:element>
6171     <xsd:element name="resumeResponse">
6172     <xsd:complexType>
6173     <xsd:sequence>
6174     <xsd:annotation>
6175     <xsd:documentation>Empty message</xsd:documentation>
6176     </xsd:annotation>
6177     </xsd:sequence>
6178     </xsd:complexType>
6179     </xsd:element>
6180
6181     <xsd:element name="batchResume">
6182     <xsd:complexType>
6183     <xsd:sequence>
6184     <xsd:element name="identifier" type="xsd:anyURI"
6185     maxOccurs="unbounded" />
6186     </xsd:sequence>
6187     </xsd:complexType>
6188     </xsd:element>
6189     <xsd:element name="batchResumeResponse">
6190     <xsd:complexType>
6191     <xsd:sequence>
6192     <xsd:element name="batchResponse" type="tBatchResponse"
6193     minOccurs="0" maxOccurs="unbounded" />
6194     </xsd:sequence>
6195     </xsd:complexType>
6196     </xsd:element>
6197
6198     <xsd:element name="complete">
6199     <xsd:complexType>
6200     <xsd:sequence>

```

```

6201 <xsd:element name="identifier" type="xsd:anyURI"/>
6202 <xsd:element name="taskData" type="xsd:anyType" minOccurs="0"/>
6203 </xsd:sequence>
6204 </xsd:complexType>
6205 </xsd:element>
6206 <xsd:element name="completeResponse">
6207 <xsd:complexType>
6208 <xsd:sequence>
6209 <xsd:annotation>
6210 <xsd:documentation>Empty message</xsd:documentation>
6211 </xsd:annotation>
6212 </xsd:sequence>
6213 </xsd:complexType>
6214 </xsd:element>
6215
6216 <xsd:element name="batchComplete">
6217 <xsd:complexType>
6218 <xsd:sequence>
6219 <xsd:element name="identifier" type="xsd:anyURI"
6220 maxOccurs="unbounded"/>
6221 </xsd:sequence>
6222 </xsd:complexType>
6223 </xsd:element>
6224 <xsd:element name="batchCompleteResponse">
6225 <xsd:complexType>
6226 <xsd:sequence>
6227 <xsd:element name="batchResponse" type="tBatchResponse"
6228 minOccurs="0" maxOccurs="unbounded"/>
6229 </xsd:sequence>
6230 </xsd:complexType>
6231 </xsd:element>
6232
6233 <xsd:element name="remove">
6234 <xsd:complexType>
6235 <xsd:sequence>
6236 <xsd:element name="identifier" type="xsd:anyURI"/>
6237 </xsd:sequence>
6238 </xsd:complexType>
6239 </xsd:element>
6240 <xsd:element name="removeResponse">
6241 <xsd:complexType>
6242 <xsd:sequence>
6243 <xsd:annotation>
6244 <xsd:documentation>Empty message</xsd:documentation>
6245 </xsd:annotation>
6246 </xsd:sequence>
6247 </xsd:complexType>
6248 </xsd:element>
6249
6250 <xsd:element name="batchRemove">
6251 <xsd:complexType>
6252 <xsd:sequence>
6253 <xsd:element name="identifier" type="xsd:anyURI"
6254 maxOccurs="unbounded"/>
6255 </xsd:sequence>
6256 </xsd:complexType>
6257 </xsd:element>
6258 <xsd:element name="batchRemoveResponse">

```

```

6259 <xsd:complexType>
6260 <xsd:sequence>
6261 <xsd:element name="batchResponse" type="tBatchResponse"
6262 minOccurs="0" maxOccurs="unbounded"/>
6263 </xsd:sequence>
6264 </xsd:complexType>
6265 </xsd:element>
6266
6267 <xsd:element name="fail">
6268 <xsd:complexType>
6269 <xsd:sequence>
6270 <xsd:element name="identifier" type="xsd:anyURI"/>
6271 <xsd:element name="fault" type="htt:tFault" minOccurs="0"/>
6272 </xsd:sequence>
6273 </xsd:complexType>
6274 </xsd:element>
6275 <xsd:element name="failResponse">
6276 <xsd:complexType>
6277 <xsd:sequence>
6278 <xsd:annotation>
6279 <xsd:documentation>Empty message</xsd:documentation>
6280 </xsd:annotation>
6281 </xsd:sequence>
6282 </xsd:complexType>
6283 </xsd:element>
6284
6285 <xsd:element name="batchFail">
6286 <xsd:complexType>
6287 <xsd:sequence>
6288 <xsd:element name="identifier" type="xsd:anyURI"
6289 maxOccurs="unbounded"/>
6290 </xsd:sequence>
6291 </xsd:complexType>
6292 </xsd:element>
6293 <xsd:element name="batchFailResponse">
6294 <xsd:complexType>
6295 <xsd:sequence>
6296 <xsd:element name="batchResponse" type="tBatchResponse"
6297 minOccurs="0" maxOccurs="unbounded"/>
6298 </xsd:sequence>
6299 </xsd:complexType>
6300 </xsd:element>
6301
6302 <xsd:element name="setPriority">
6303 <xsd:complexType>
6304 <xsd:sequence>
6305 <xsd:element name="identifier" type="xsd:anyURI"/>
6306 <xsd:element name="priority" type="htt:tPriority"/>
6307 </xsd:sequence>
6308 </xsd:complexType>
6309 </xsd:element>
6310 <xsd:element name="setPriorityResponse">
6311 <xsd:complexType>
6312 <xsd:sequence>
6313 <xsd:annotation>
6314 <xsd:documentation>Empty message</xsd:documentation>
6315 </xsd:annotation>
6316 </xsd:sequence>

```

```

6317       </xsd:complexType>
6318     </xsd:element>
6319
6320     <xsd:element name="batchSetPriority">
6321       <xsd:complexType>
6322         <xsd:sequence>
6323           <xsd:element name="identifier" type="xsd:anyURI"
6324 maxOccurs="unbounded"/>
6325           <xsd:element name="priority" type="http:tPriority"/>
6326         </xsd:sequence>
6327       </xsd:complexType>
6328     </xsd:element>
6329     <xsd:element name="batchSetPriorityResponse">
6330       <xsd:complexType>
6331         <xsd:sequence>
6332           <xsd:element name="batchResponse" type="tBatchResponse"
6333 minOccurs="0" maxOccurs="unbounded"/>
6334         </xsd:sequence>
6335       </xsd:complexType>
6336     </xsd:element>
6337
6338     <xsd:element name="addAttachment">
6339       <xsd:complexType>
6340         <xsd:sequence>
6341           <xsd:element name="taskIdIdentifier" type="xsd:anyURI"/>
6342           <xsd:element name="name" type="xsd:string"/>
6343           <xsd:element name="accessType" type="xsd:string"/>
6344           <xsd:element name="contentType" type="xsd:string"/>
6345           <xsd:element name="attachment" type="xsd:anyType"/>
6346         </xsd:sequence>
6347       </xsd:complexType>
6348     </xsd:element>
6349     <xsd:element name="addAttachmentResponse">
6350       <xsd:complexType>
6351         <xsd:sequence>
6352           <xsd:element name="identifier" type="xsd:anyURI"/>
6353         </xsd:sequence>
6354       </xsd:complexType>
6355     </xsd:element>
6356
6357     <xsd:element name="getAttachmentInfos">
6358       <xsd:complexType>
6359         <xsd:sequence>
6360           <xsd:element name="identifier" type="xsd:anyURI"/>
6361         </xsd:sequence>
6362       </xsd:complexType>
6363     </xsd:element>
6364     <xsd:element name="getAttachmentInfosResponse">
6365       <xsd:complexType>
6366         <xsd:sequence>
6367           <xsd:element name="info" type="http:tAttachmentInfo" minOccurs="0"
6368 maxOccurs="unbounded"/>
6369         </xsd:sequence>
6370       </xsd:complexType>
6371     </xsd:element>
6372
6373     <xsd:element name="getAttachment">
6374       <xsd:complexType>

```



```

6375 <del><xsd:sequence>
6376 <del><xsd:element name="taskIdentifier" type="xsd:anyURI"/>
6377 <del><xsd:element name="attachmentIdentifier" type="xsd:anyURI"/>
6378 <del></xsd:sequence>
6379 <del></xsd:complexType>
6380 <del></xsd:element>
6381 <del><xsd:element name="getAttachmentResponse">
6382 <del><xsd:complexType>
6383 <del><xsd:sequence>
6384 <del><xsd:element name="attachment" type="http:tAttachment"
6385 minOccurs="0" maxOccurs="unbounded"/>
6386 <del></xsd:sequence>
6387 <del></xsd:complexType>
6388 <del></xsd:element>
6389
6390 <del><xsd:element name="deleteAttachment">
6391 <del><xsd:complexType>
6392 <del><xsd:sequence>
6393 <del><xsd:element name="taskIdentifier" type="xsd:anyURI"/>
6394 <del><xsd:element name="attachmentIdentifier" type="xsd:anyURI"/>
6395 <del></xsd:sequence>
6396 <del></xsd:complexType>
6397 <del></xsd:element>
6398 <del><xsd:element name="deleteAttachmentResponse">
6399 <del><xsd:complexType>
6400 <del><xsd:sequence>
6401 <del><xsd:annotation>
6402 <del><xsd:documentation>Empty message</xsd:documentation>
6403 <del></xsd:annotation>
6404 <del></xsd:sequence>
6405 <del></xsd:complexType>
6406 <del></xsd:element>
6407
6408 <del><xsd:element name="addComment">
6409 <del><xsd:complexType>
6410 <del><xsd:sequence>
6411 <del><xsd:element name="identifier" type="xsd:anyURI"/>
6412 <del><xsd:element name="text" type="xsd:string"/>
6413 <del></xsd:sequence>
6414 <del></xsd:complexType>
6415 <del></xsd:element>
6416 <del><xsd:element name="addCommentResponse">
6417 <del><xsd:complexType>
6418 <del><xsd:sequence>
6419 <del><xsd:element name="commentID" type="xsd:string"/>
6420 <del></xsd:sequence>
6421 <del></xsd:complexType>
6422 <del></xsd:element>
6423
6424 <xsd:element name="updateComment">
6425 <xsd:complexType>
6426 <xsd:sequence>
6427 <xsd:element name="taskIdentifier" type="xsd:anyURI"/>
6428 <xsd:element name="commentIdentifier" type="xsd:anyURI"/>
6429 <xsd:element name="text" type="xsd:string"/>
6430 </xsd:sequence>
6431 </xsd:complexType>
6432 </xsd:element>

```

```

6433 <xsd:element name="updateCommentResponse" >
6434 <xsd:complexType>
6435 <xsd:sequence>
6436 <xsd:annotation>
6437 <xsd:documentation>Empty message</xsd:documentation>
6438 </xsd:annotation>
6439 </xsd:sequence>
6440 </xsd:complexType>
6441 </xsd:element>
6442
6443 <xsd:element name="deleteComment">
6444 <xsd:complexType>
6445 <xsd:sequence>
6446 <xsd:element name="taskIdIdentifier" type="xsd:anyURI"/>
6447 <xsd:element name="commentIdentifier" type="xsd:anyURI"/>
6448 </xsd:sequence>
6449 </xsd:complexType>
6450 </xsd:element>
6451 <xsd:element name="deleteCommentResponse">
6452 <xsd:complexType>
6453 <xsd:sequence>
6454 <xsd:annotation>
6455 <xsd:documentation>Empty message</xsd:documentation>
6456 </xsd:annotation>
6457 </xsd:sequence>
6458 </xsd:complexType>
6459 </xsd:element>
6460
6461 <xsd:element name="getComments">
6462 <xsd:complexType>
6463 <xsd:sequence>
6464 <xsd:element name="identifier" type="xsd:anyURI"/>
6465 </xsd:sequence>
6466 </xsd:complexType>
6467 </xsd:element>
6468 <xsd:element name="getCommentsResponse">
6469 <xsd:complexType>
6470 <xsd:sequence>
6471 <xsd:element name="comment" type="htt:tComment" minOccurs="0"
6472 maxOccurs="unbounded"/>
6473 </xsd:sequence>
6474 </xsd:complexType>
6475 </xsd:element>
6476
6477 <xsd:element name="skip">
6478 <xsd:complexType>
6479 <xsd:sequence>
6480 <xsd:element name="identifier" type="xsd:anyURI"/>
6481 </xsd:sequence>
6482 </xsd:complexType>
6483 </xsd:element>
6484 <xsd:element name="skipResponse">
6485 <xsd:complexType>
6486 <xsd:sequence>
6487 <xsd:annotation>
6488 <xsd:documentation>Empty message</xsd:documentation>
6489 </xsd:annotation>
6490 </xsd:sequence>

```

```

6491 </xsd:complexType>
6492 </xsd:element>
6493
6494 <xsd:element name="batchSkip">
6495 <xsd:complexType>
6496 <xsd:sequence>
6497 <xsd:element name="identifier" type="xsd:anyURI"
6498 maxOccurs="unbounded"/>
6499 </xsd:sequence>
6500 </xsd:complexType>
6501 </xsd:element>
6502 <xsd:element name="batchSkipResponse">
6503 <xsd:complexType>
6504 <xsd:sequence>
6505 <xsd:element name="batchResponse" type="tBatchResponse"
6506 minOccurs="0" maxOccurs="unbounded"/>
6507 </xsd:sequence>
6508 </xsd:complexType>
6509 </xsd:element>
6510
6511 <xsd:element name="forward">
6512 <xsd:complexType>
6513 <xsd:sequence>
6514 <xsd:element name="identifier" type="xsd:anyURI"/>
6515 <xsd:element name="organizationalEntity"
6516 type="htt:tOrganizationalEntity"/>
6517 </xsd:sequence>
6518 </xsd:complexType>
6519 </xsd:element>
6520 <xsd:element name="forwardResponse">
6521 <xsd:complexType>
6522 <xsd:sequence>
6523 <xsd:annotation>
6524 <xsd:documentation>Empty message</xsd:documentation>
6525 </xsd:annotation>
6526 </xsd:sequence>
6527 </xsd:complexType>
6528 </xsd:element>
6529
6530 <xsd:element name="batchForward">
6531 <xsd:complexType>
6532 <xsd:sequence>
6533 <xsd:element name="identifier" type="xsd:anyURI"
6534 maxOccurs="unbounded"/>
6535 <xsd:element name="organizationalEntity"
6536 type="htt:tOrganizationalEntity"/>
6537 </xsd:sequence>
6538 </xsd:complexType>
6539 </xsd:element>
6540 <xsd:element name="batchForwardResponse">
6541 <xsd:complexType>
6542 <xsd:sequence>
6543 <xsd:element name="batchResponse" type="tBatchResponse"
6544 minOccurs="0" maxOccurs="unbounded"/>
6545 </xsd:sequence>
6546 </xsd:complexType>
6547 </xsd:element>
6548

```

```

6549 <del><xsd:element name="delegate">
6550 <del><xsd:complexType>
6551 <del><xsd:sequence>
6552 <del><xsd:element name="identifier" type="xsd:anyURI"/>
6553 <del><xsd:element name="organizationalEntity"
6554 type="htt:tOrganizationalEntity"/>
6555 <del></xsd:sequence>
6556 <del></xsd:complexType>
6557 <del></xsd:element>
6558 <del><xsd:element name="delegateResponse">
6559 <del><xsd:complexType>
6560 <del><xsd:sequence>
6561 <del><xsd:annotation>
6562 <del><xsd:documentation>Empty message</xsd:documentation>
6563 <del></xsd:annotation>
6564 <del></xsd:sequence>
6565 <del></xsd:complexType>
6566 <del></xsd:element>
6567
6568 <del><xsd:element name="batchDelegate">
6569 <del><xsd:complexType>
6570 <del><xsd:sequence>
6571 <del><xsd:element name="identifier" type="xsd:anyURI"
6572 minOccurs="unbounded"/>
6573 <del><xsd:element name="organizationalEntity"
6574 type="htt:tOrganizationalEntity"/>
6575 <del></xsd:sequence>
6576 <del></xsd:complexType>
6577 <del></xsd:element>
6578 <del><xsd:element name="batchDelegateResponse">
6579 <del><xsd:complexType>
6580 <del><xsd:sequence>
6581 <del><xsd:element name="batchResponse" type="tBatchResponse"
6582 minOccurs="0" maxOccurs="unbounded"/>
6583 <del></xsd:sequence>
6584 <del></xsd:complexType>
6585 <del></xsd:element>
6586
6587 <del><xsd:element name="getRendering">
6588 <del><xsd:complexType>
6589 <del><xsd:sequence>
6590 <del><xsd:element name="identifier" type="xsd:anyType"/>
6591 <del><xsd:element name="renderingType" type="xsd:QName"/>
6592 <del></xsd:sequence>
6593 <del></xsd:complexType>
6594 <del></xsd:element>
6595 <del><xsd:element name="getRenderingResponse">
6596 <del><xsd:complexType>
6597 <del><xsd:sequence>
6598 <del><xsd:element name="rendering" type="xsd:anyType"/>
6599 <del></xsd:sequence>
6600 <del></xsd:complexType>
6601 <del></xsd:element>
6602
6603 <del><xsd:element name="getRenderingTypes">
6604 <del><xsd:complexType>
6605 <del><xsd:sequence>
6606 <del><xsd:element name="identifier" type="xsd:anyType"/>

```

```

6607 </xsd:sequence>
6608 </xsd:complexType>
6609 </xsd:element>
6610 <xsd:element name="getRenderingTypesResponse">
6611 <xsd:complexType>
6612 <xsd:sequence>
6613 <xsd:element name="renderingType" type="xsd:QName" minOccurs="0"
6614 maxOccurs="unbounded"/>
6615 </xsd:sequence>
6616 </xsd:complexType>
6617 </xsd:element>
6618
6619 <xsd:element name="getTaskDetails">
6620 <xsd:complexType>
6621 <xsd:sequence>
6622 <xsd:element name="identifier" type="xsd:anyURI"/>
6623 </xsd:sequence>
6624 </xsd:complexType>
6625 </xsd:element>
6626 <xsd:element name="getTaskDetailsResponse">
6627 <xsd:complexType>
6628 <xsd:sequence>
6629 <xsd:element name="taskDetails" type="htt:tTaskDetails"/>
6630 </xsd:sequence>
6631 </xsd:complexType>
6632 </xsd:element>
6633
6634 <xsd:element name="getTaskDescription">
6635 <xsd:complexType>
6636 <xsd:sequence>
6637 <xsd:element name="identifier" type="xsd:anyURI"/>
6638 <xsd:element name="contentType" type="xsd:string" minOccurs="0"/>
6639 </xsd:sequence>
6640 </xsd:complexType>
6641 </xsd:element>
6642 <xsd:element name="getTaskDescriptionResponse">
6643 <xsd:complexType>
6644 <xsd:sequence>
6645 <xsd:element name="description" type="xsd:string"/>
6646 </xsd:sequence>
6647 </xsd:complexType>
6648 </xsd:element>
6649
6650 <xsd:element name="setOutput">
6651 <xsd:complexType>
6652 <xsd:sequence>
6653 <xsd:element name="identifier" type="xsd:anyURI"/>
6654 <xsd:element name="part" type="xsd:NCName" minOccurs="0"/>
6655 <xsd:element name="taskData" type="xsd:anyType"/>
6656 </xsd:sequence>
6657 </xsd:complexType>
6658 </xsd:element>
6659 <xsd:element name="setOutputResponse">
6660 <xsd:complexType>
6661 <xsd:sequence>
6662 <xsd:annotation>
6663 <xsd:documentation>Empty message</xsd:documentation>
6664 </xsd:annotation>

```

```

6665 </xsd:sequence>
6666 </xsd:complexType>
6667 </xsd:element>
6668
6669 <xsd:element name="deleteOutput">
6670 <xsd:complexType>
6671 <xsd:sequence>
6672 <xsd:element name="identifier" type="xsd:anyURI"/>
6673 </xsd:sequence>
6674 </xsd:complexType>
6675 </xsd:element>
6676 <xsd:element name="deleteOutputResponse">
6677 <xsd:complexType>
6678 <xsd:sequence>
6679 <xsd:annotation>
6680 <xsd:documentation>Empty message</xsd:documentation>
6681 </xsd:annotation>
6682 </xsd:sequence>
6683 </xsd:complexType>
6684 </xsd:element>
6685
6686 <xsd:element name="setFault">
6687 <xsd:complexType>
6688 <xsd:sequence>
6689 <xsd:element name="identifier" type="xsd:anyURI"/>
6690 <xsd:element name="fault" type="http:Fault"/>
6691 </xsd:sequence>
6692 </xsd:complexType>
6693 </xsd:element>
6694 <xsd:element name="setFaultResponse">
6695 <xsd:complexType>
6696 <xsd:sequence>
6697 <xsd:annotation>
6698 <xsd:documentation>Empty message</xsd:documentation>
6699 </xsd:annotation>
6700 </xsd:sequence>
6701 </xsd:complexType>
6702 </xsd:element>
6703
6704 <xsd:element name="deleteFault">
6705 <xsd:complexType>
6706 <xsd:sequence>
6707 <xsd:element name="identifier" type="xsd:anyURI"/>
6708 </xsd:sequence>
6709 </xsd:complexType>
6710 </xsd:element>
6711 <xsd:element name="deleteFaultResponse">
6712 <xsd:complexType>
6713 <xsd:sequence>
6714 <xsd:annotation>
6715 <xsd:documentation>Empty message</xsd:documentation>
6716 </xsd:annotation>
6717 </xsd:sequence>
6718 </xsd:complexType>
6719 </xsd:element>
6720
6721 <xsd:element name="getInput">
6722 <xsd:complexType>

```

```

6723 <xsd:sequence>
6724 <xsd:element name="identifier" type="xsd:anyURI"/>
6725 <xsd:element name="part" type="xsd:NCName" minOccurs="0"/>
6726 </xsd:sequence>
6727 </xsd:complexType>
6728 </xsd:element>
6729 <xsd:element name="getInputResponse">
6730 <xsd:complexType>
6731 <xsd:sequence>
6732 <xsd:element name="taskData" type="xsd:anyType"/>
6733 </xsd:sequence>
6734 </xsd:complexType>
6735 </xsd:element>
6736
6737 <xsd:element name="getOutput">
6738 <xsd:complexType>
6739 <xsd:sequence>
6740 <xsd:element name="identifier" type="xsd:anyURI"/>
6741 <xsd:element name="part" type="xsd:NCName" minOccurs="0"/>
6742 </xsd:sequence>
6743 </xsd:complexType>
6744 </xsd:element>
6745 <xsd:element name="getOutputResponse">
6746 <xsd:complexType>
6747 <xsd:sequence>
6748 <xsd:element name="taskData" type="xsd:anyType"/>
6749 </xsd:sequence>
6750 </xsd:complexType>
6751 </xsd:element>
6752
6753 <xsd:element name="getFault">
6754 <xsd:complexType>
6755 <xsd:sequence>
6756 <xsd:element name="identifier" type="xsd:anyURI"/>
6757 </xsd:sequence>
6758 </xsd:complexType>
6759 </xsd:element>
6760 <xsd:element name="getFaultResponse">
6761 <xsd:complexType>
6762 <xsd:sequence>
6763 <xsd:element name="fault" type="htt:tFault"/>
6764 </xsd:sequence>
6765 </xsd:complexType>
6766 </xsd:element>
6767
6768 <xsd:element name="getMyTaskAbstracts">
6769 <xsd:complexType>
6770 <xsd:sequence>
6771 <xsd:element name="taskType" type="xsd:string"/>
6772 <xsd:element name="genericHumanRole" type="xsd:string"
6773 minOccurs="0"/>
6774 <xsd:element name="workQueue" type="xsd:string" minOccurs="0"/>
6775 <xsd:element name="status" type="htt:tStatus" minOccurs="0"
6776 maxOccurs="unbounded"/>
6777 <xsd:element name="whereClause" type="xsd:string" minOccurs="0"/>
6778 <xsd:element name="createdOnClause" type="xsd:string"
6779 minOccurs="0"/>
6780 <xsd:element name="maxTasks" type="xsd:int" minOccurs="0"/>

```

```

6781     <xsd:element name="orderByClause" type="xsd:string"
6782 minOccurs="0"/>
6783     <xsd:element name="createdOnClause" type="xsd:string"
6784 minOccurs="0"/>
6785     <xsd:element name="maxTasks" type="xsd:int" minOccurs="0"/>
6786     <xsd:element name="taskIndexOffset" type="xsd:int"
6787 minOccurs="0"/>
6788     </xsd:sequence>
6789   </xsd:complexType>
6790 </xsd:element>
6791 <xsd:element name="getMyTaskAbstractsResponse">
6792   <xsd:complexType>
6793     <xsd:sequence>
6794       <xsd:element name="taskAbstract" type="htt:tTaskAbstract"
6795 minOccurs="0" maxOccurs="unbounded"/>
6796     </xsd:sequence>
6797   </xsd:complexType>
6798 </xsd:element>
6799
6800   <xsd:element name="getMyTaskDetails">
6801     <xsd:complexType>
6802       <xsd:sequence>
6803         <xsd:element name="taskType" type="xsd:string"/>
6804         <xsd:element name="genericHumanRole" type="xsd:string"
6805 minOccurs="0"/>
6806         <xsd:element name="workQueue" type="xsd:string" minOccurs="0"/>
6807         <xsd:element name="status" type="htt:tStatus" minOccurs="0"
6808 maxOccurs="unbounded"/>
6809         <xsd:element name="whereClause" type="xsd:string" minOccurs="0"/>
6810         <xsd:element name="createdOnClauseorderByClause"
6811 type="xsd:string" minOccurs="0"/>
6812         <xsd:element name="createdOnClause" type="xsd:string"
6813 minOccurs="0"/>
6814         <xsd:element name="maxTasks" type="xsd:int" minOccurs="0"/>
6815         <xsd:element name="maxTaskstaskIndexOffset" type="xsd:int"
6816 minOccurs="0"/>
6817       </xsd:sequence>
6818     </xsd:complexType>
6819   </xsd:element>
6820 <xsd:element name="getMyTaskDetailsResponse">
6821   <xsd:complexType>
6822     <xsd:sequence>
6823       <xsd:element name="taskDetails" type="htt:tTaskDetails"
6824 minOccurs="0" maxOccurs="unbounded"/>
6825     </xsd:sequence>
6826   </xsd:complexType>
6827 </xsd:element>
6828
6829   <xsd:element name="query">
6830     <xsd:complexType>
6831       <xsd:sequence>
6832         <xsd:element name="selectClause" type="xsd:string"/>
6833         <xsd:element name="whereClause" type="xsd:string" minOccurs="0"/>
6834         <xsd:element name="orderByClause" type="xsd:string"
6835 minOccurs="0"/>
6836         <xsd:element name="maxTasks" type="xsd:int" minOccurs="0"/>
6837         <xsd:element name="taskIndexOffset" type="xsd:int"
6838 minOccurs="0"/>

```



```

6839     </xsd:sequence>
6840   </xsd:complexType>
6841 </xsd:element>
6842 <xsd:element name="queryResponse">
6843   <xsd:complexType>
6844     <xsd:sequence>
6845       <xsd:element name="taskQueryResultSet"
6846 type="htt:tTaskQueryResultSet" />
6847     </xsd:sequence>
6848   </xsd:complexType>
6849 </xsd:element>
6850
6851 <xsd:element name="activate">
6852   <xsd:complexType>
6853     <xsd:sequence>
6854       <xsd:element name="identifier" type="xsd:anyURI" />
6855     </xsd:sequence>
6856   </xsd:complexType>
6857 </xsd:element>
6858 <xsd:element name="activateResponse">
6859   <xsd:complexType>
6860     <xsd:sequence>
6861       <xsd:annotation>
6862         <xsd:documentation>Empty message</xsd:documentation>
6863       </xsd:annotation>
6864     </xsd:sequence>
6865   </xsd:complexType>
6866 </xsd:element>
6867
6868 <xsd:element name="batchActivate">
6869   <xsd:complexType>
6870     <xsd:sequence>
6871       <xsd:element name="identifier" type="xsd:anyURI"
6872 maxOccurs="unbounded" />
6873     </xsd:sequence>
6874   </xsd:complexType>
6875 </xsd:element>
6876 <xsd:element name="batchActivateResponse">
6877   <xsd:complexType>
6878     <xsd:sequence>
6879       <xsd:element name="batchResponse" type="tBatchResponse"
6880 minOccurs="0" maxOccurs="unbounded" />
6881     </xsd:sequence>
6882   </xsd:complexType>
6883 </xsd:element>
6884
6885 <xsd:element name="nominate">
6886   <xsd:complexType>
6887     <xsd:sequence>
6888       <xsd:element name="identifier" type="xsd:anyURI" />
6889       <xsd:element name="organizationalEntity"
6890 type="htt:tOrganizationalEntity" />
6891     </xsd:sequence>
6892   </xsd:complexType>
6893 </xsd:element>
6894 <xsd:element name="nominateResponse">
6895   <xsd:complexType>
6896     <xsd:sequence>

```

```

6897         <xsd:annotation>
6898             <xsd:documentation>Empty message</xsd:documentation>
6899         </xsd:annotation>
6900     </xsd:sequence>
6901 </xsd:complexType>
6902 </xsd:element>
6903
6904     <xsd:element name="batchNominate">
6905         <xsd:complexType>
6906             <xsd:sequence>
6907                 <xsd:element name="identifier" type="xsd:anyURI"
6908 maxOccurs="unbounded" />
6909             </xsd:sequence>
6910         </xsd:complexType>
6911     </xsd:element>
6912     <xsd:element name="batchNominateResponse">
6913         <xsd:complexType>
6914             <xsd:sequence>
6915                 <xsd:element name="batchResponse" type="tBatchResponse"
6916 minOccurs="0" maxOccurs="unbounded" />
6917             </xsd:sequence>
6918         </xsd:complexType>
6919     </xsd:element>
6920
6921     <xsd:element name="setGenericHumanRole">
6922         <xsd:complexType>
6923             <xsd:sequence>
6924                 <xsd:element name="identifier" type="xsd:anyURI" />
6925                 <xsd:element name="genericHumanRole" type="xsd:string" />
6926                 <xsd:element name="organizationalEntity"
6927 type="htt:tOrganizationalEntity" />
6928             </xsd:sequence>
6929         </xsd:complexType>
6930     </xsd:element>
6931     <xsd:element name="setGenericHumanRoleResponse">
6932         <xsd:complexType>
6933             <xsd:sequence>
6934                 <xsd:annotation>
6935                     <xsd:documentation>Empty message</xsd:documentation>
6936                 </xsd:annotation>
6937             </xsd:sequence>
6938         </xsd:complexType>
6939     </xsd:element>
6940
6941     <xsd:element name="batchSetGenericHumanRole">
6942         <xsd:complexType>
6943             <xsd:sequence>
6944                 <xsd:element name="identifier" type="xsd:anyURI"
6945 maxOccurs="unbounded" />
6946                 <xsd:element name="genericHumanRole" type="xsd:string" />
6947                 <xsd:element name="organizationalEntity"
6948 type="htt:tOrganizationalEntity" />
6949             </xsd:sequence>
6950         </xsd:complexType>
6951     </xsd:element>
6952     <xsd:element name="batchSetGenericHumanRoleResponse">
6953         <xsd:complexType>
6954             <xsd:sequence>

```

```

6955         <xsd:element name="batchResponse" type="tBatchResponse"
6956 minOccurs="0" maxOccurs="unbounded" />
6957     </xsd:sequence>
6958 </xsd:complexType>
6959 </xsd:element>
6960
6961
6962 <xsd:element name="getOutcome">
6963 <xsd:complexType>
6964 <xsd:sequence>
6965 <xsd:element name="identifier" type="xsd:anyURI" />
6966 </xsd:sequence>
6967 </xsd:complexType>
6968 </xsd:element>
6969 <xsd:element name="getOutcomeResponse">
6970 <xsd:complexType>
6971 <xsd:sequence>
6972 <xsd:element name="outcome" type="xsd:string" />
6973 </xsd:sequence>
6974 </xsd:complexType>
6975 </xsd:element>
6976
6977 <xsd:element name="getTaskOperations">
6978 <xsd:complexType>
6979 <xsd:sequence>
6980 <xsd:element name="identifier" type="xsd:anyURI" />
6981 </xsd:sequence>
6982 </xsd:complexType>
6983 </xsd:element>
6984 <xsd:element name="getTaskOperationsResponse">
6985 <xsd:complexType>
6986 <xsd:sequence>
6987 <xsd:element name="taskOperations" type="htt:tTaskOperations" />
6988 </xsd:sequence>
6989 </xsd:complexType>
6990 </xsd:element>
6991
6992 <xsd:element name="getTaskInstanceData">
6993 <xsd:complexType>
6994 <xsd:sequence>
6995 <xsd:element name="identifier" type="xsd:anyURI" />
6996 <xsd:element name="properties" type="xsd:string" />
6997 <xsd:element name="renderingPreferences"
6998 type="htt:tRenderingTypes" minOccurs="0" maxOccurs="unbounded" />
6999 </xsd:sequence>
7000 </xsd:complexType>
7001 </xsd:element>
7002 <xsd:element name="getTaskInstanceDataResponse">
7003 <xsd:complexType>
7004 <xsd:sequence>
7005 <xsd:element name="taskInstanceData"
7006 type="htt:tTaskInstanceData" />
7007 </xsd:sequence>
7008 </xsd:complexType>
7009 </xsd:element>
7010
7011 <xsd:element name="getTaskHistory">
7012 <xsd:complexType>

```

```

7013 <del><xsd:sequence>
7014 <del>  <xsd:element name="identifier" type="xsd:anyURI"/>
7015 <del>  <xsd:element name="filter" type="htt:tTaskHistoryFilter"
7016 <del>minOccurs="0"/>
7017 <del>  <xsd:element name="startIndex" type="xsd:integer" minOccurs="0"/>
7018 <del>  <xsd:element name="maxTasks" type="xsd:integer" minOccurs="0"/>
7019 <del></xsd:sequence>
7020 <del>  <xsd:attribute name="includeData" type="xsd:boolean"/>
7021 <del></xsd:complexType>
7022 <del></xsd:element>
7023 <del><xsd:element name="getTaskHistoryResponse">
7024 <del>  <xsd:complexType>
7025 <del>    <xsd:sequence>
7026 <del>      <xsd:element name="taskEvent" type="htt:tTaskEventType"
7027 <del>minOccurs="0" maxOccurs="unbounded"/>
7028 <del>    </xsd:sequence>
7029 <del>  </xsd:complexType>
7030 <del></xsd:element>
7031
7032 <del><xsd:element name="setTaskStartDeadlineExpression">
7033 <del>  <xsd:complexType>
7034 <del>    <xsd:sequence>
7035 <del>      <xsd:element name="identifier" type="xsd:anyURI"/>
7036 <del>      <xsd:element name="deadlineName" type="xsd:NCName"/>
7037 <del>      <xsd:element name="deadlineExpression" type="xsd:string"/>
7038 <del>    </xsd:sequence>
7039 <del>  </xsd:complexType>
7040 <del></xsd:element>
7041 <del><xsd:element name="setTaskStartDeadlineExpressionResponse">
7042 <del>  <xsd:complexType>
7043 <del>    <xsd:sequence>
7044 <del>      <xsd:annotation>
7045 <del>        <xsd:documentation>Empty message</xsd:documentation>
7046 <del>      </xsd:annotation>
7047 <del>    </xsd:sequence>
7048 <del>  </xsd:complexType>
7049 <del></xsd:element>
7050
7051 <del><xsd:element name="setTaskStartDurationExpression">
7052 <del>  <xsd:complexType>
7053 <del>    <xsd:sequence>
7054 <del>      <xsd:element name="identifier" type="xsd:anyURI"/>
7055 <del>      <xsd:element name="deadlineName" type="xsd:NCName"/>
7056 <del>      <xsd:element name="durationExpression" type="xsd:string"/>
7057 <del>    </xsd:sequence>
7058 <del>  </xsd:complexType>
7059 <del></xsd:element>
7060 <del><xsd:element name="setTaskStartDurationExpressionResponse">
7061 <del>  <xsd:complexType>
7062 <del>    <xsd:sequence>
7063 <del>      <xsd:annotation>
7064 <del>        <xsd:documentation>Empty message</xsd:documentation>
7065 <del>      </xsd:annotation>
7066 <del>    </xsd:sequence>
7067 <del>  </xsd:complexType>
7068 <del></xsd:element>
7069
7070 <del><xsd:element name="setTaskCompletionDeadlineExpression">

```

```

7071 </xsd:complexType>
7072 </xsd:sequence>
7073 </xsd:element name="identifier" type="xsd:anyURI"/>
7074 </xsd:element name="deadlineName" type="xsd:NCName"/>
7075 </xsd:element name="deadlineExpression" type="xsd:string"/>
7076 </xsd:sequence>
7077 </xsd:complexType>
7078 </xsd:element>
7079 </xsd:element name="setTaskCompletionDeadlineExpressionResponse">
7080 </xsd:complexType>
7081 </xsd:sequence>
7082 </xsd:annotation>
7083 </xsd:documentation>Empty message</xsd:documentation>
7084 </xsd:annotation>
7085 </xsd:sequence>
7086 </xsd:complexType>
7087 </xsd:element>
7088
7089 </xsd:element name="setTaskCompletionDurationExpression">
7090 </xsd:complexType>
7091 </xsd:sequence>
7092 </xsd:element name="identifier" type="xsd:anyURI"/>
7093 </xsd:element name="deadlineName" type="xsd:NCName"/>
7094 </xsd:element name="durationExpression" type="xsd:string"/>
7095 </xsd:sequence>
7096 </xsd:complexType>
7097 </xsd:element>
7098 </xsd:element name="setTaskCompletionDurationExpressionResponse">
7099 </xsd:complexType>
7100 </xsd:sequence>
7101 </xsd:annotation>
7102 </xsd:documentation>Empty message</xsd:documentation>
7103 </xsd:annotation>
7104 </xsd:sequence>
7105 </xsd:complexType>
7106 </xsd:element>
7107
7108 <!-- Fault elements -->
7109 </xsd:element name="illegalState">
7110 </xsd:complexType>
7111 </xsd:sequence>
7112 </xsd:element name="status" type="htt:tStatus"/>
7113 </xsd:element name="message" type="xsd:string"/>
7114 </xsd:sequence>
7115 </xsd:complexType>
7116 </xsd:element>
7117
7118 </xsd:element name="illegalArgument" type="xsd:string"/>
7119
7120 </xsd:element name="illegalAccess" type="xsd:string"/>
7121
7122 </xsd:element name="illegalOperation" type="xsd:string"/>
7123
7124 </xsd:element name="recipientNotAllowed" type="xsd:string"/>
7125
7126 </xsd:complexType name="tBatchResponse">
7127 </xsd:sequence>
7128 </xsd:element name="identifier" type="xsd:anyURI"/>

```

```

7129     <xsd:choice>
7130         <xsd:element ref="illegalState" />
7131         <xsd:element ref="illegalArgument" />
7132         <xsd:element ref="illegalAccess" />
7133         <xsd:element ref="illegalOperation" />
7134         <xsd:element ref="recipientNotAllowed" />
7135         <xsd:any namespace="##other" processContents="lax" />
7136     </xsd:choice>
7137 </xsd:sequence>
7138 </xsd:complexType>
7139
7140 </xsd:schema>
7141 </wsdl:types>
7142
7143 <!-- Declaration of messages -->
7144 <wsdl:message name="addAttachment">
7145     <wsdl:part name="addAttachment" element="addAttachment" />
7146 </wsdl:message>
7147 <wsdl:message name="addAttachmentResponse">
7148     <wsdl:part name="addAttachmentResponse" element="addAttachmentResponse" />
7149 </wsdl:message>
7150
7151 <wsdl:message name="addComment">
7152     <wsdl:part name="addComment" element="addComment" />
7153 </wsdl:message>
7154 <wsdl:message name="addCommentResponse">
7155     <wsdl:part name="addCommentResponse" element="addCommentResponse" />
7156 </wsdl:message>
7157
7158 <wsdl:message name="claim">
7159     <wsdl:part name="claim" element="claim" />
7160 </wsdl:message>
7161 <wsdl:message name="claimResponse">
7162     <wsdl:part name="claimResponse" element="claimResponse" />
7163 </wsdl:message>
7164
7165 <wsdl:message name="batchClaim">
7166     <wsdl:part name="batchClaim" element="batchClaim" />
7167 </wsdl:message>
7168 <wsdl:message name="batchClaimResponse">
7169     <wsdl:part name="batchClaimResponse" element="batchClaimResponse" />
7170 </wsdl:message>
7171
7172 <wsdl:message name="complete">
7173     <wsdl:part name="complete" element="complete" />
7174 </wsdl:message>
7175 <wsdl:message name="completeResponse">
7176     <wsdl:part name="completeResponse" element="completeResponse" />
7177 </wsdl:message>
7178
7179 <wsdl:message name="batchComplete">
7180     <wsdl:part name="batchComplete" element="batchComplete" />
7181 </wsdl:message>
7182 <wsdl:message name="batchCompleteResponse">
7183     <wsdl:part name="batchCompleteResponse" element="batchCompleteResponse" />
7184 </wsdl:message>
7185
7186 <wsdl:message name="delegate">

```

```

7187     <wsdl:part name="delegate" element="delegate" />
7188 </wsdl:message>
7189 <wsdl:message name="delegateResponse">
7190     <wsdl:part name="delegateResponse" element="delegateResponse" />
7191 </wsdl:message>
7192
7193 <wsdl:message name="batchDelegate">
7194     <wsdl:part name="batchDelegate" element="batchDelegate" />
7195 </wsdl:message>
7196 <wsdl:message name="batchDelegateResponse">
7197     <wsdl:part name="batchDelegateResponse" element="batchDelegateResponse" />
7198 </wsdl:message>
7199
7200 <wsdl:message name="deleteAttachment">
7201     <wsdl:part name="deleteAttachment" element="deleteAttachment" />
7202 </wsdl:message>
7203 <wsdl:message name="deleteAttachmentResponse">
7204     <wsdl:part name="deleteAttachmentResponse"
7205 element="deleteAttachmentResponse" />
7206 </wsdl:message>
7207
7208 <wsdl:message name="deleteComment">
7209     <wsdl:part name="deleteComment" element="deleteComment" />
7210 </wsdl:message>
7211 <wsdl:message name="deleteCommentResponse">
7212     <wsdl:part name="deleteCommentResponse" element="deleteCommentResponse" />
7213 </wsdl:message>
7214
7215 <wsdl:message name="deleteFault">
7216     <wsdl:part name="deleteFault" element="deleteFault" />
7217 </wsdl:message>
7218 <wsdl:message name="deleteFaultResponse">
7219     <wsdl:part name="deleteFaultResponse" element="deleteFaultResponse" />
7220 </wsdl:message>
7221
7222 <wsdl:message name="deleteOutput">
7223     <wsdl:part name="deleteOutput" element="deleteOutput" />
7224 </wsdl:message>
7225 <wsdl:message name="deleteOutputResponse">
7226     <wsdl:part name="deleteOutputResponse" element="deleteOutputResponse" />
7227 </wsdl:message>
7228
7229 <wsdl:message name="fail">
7230     <wsdl:part name="fail" element="fail" />
7231 </wsdl:message>
7232 <wsdl:message name="failResponse">
7233     <wsdl:part name="failResponse" element="failResponse" />
7234 </wsdl:message>
7235
7236 <wsdl:message name="batchFail">
7237     <wsdl:part name="batchFail" element="batchFail" />
7238 </wsdl:message>
7239 <wsdl:message name="batchFailResponse">
7240     <wsdl:part name="batchFailResponse" element="batchFailResponse" />
7241 </wsdl:message>
7242
7243 <wsdl:message name="forward">
7244     <wsdl:part name="forward" element="forward" />

```

```

7245 </wsdl:message>
7246 <wsdl:message name="forwardResponse">
7247   <wsdl:part name="forwardResponse" element="forwardResponse" />
7248 </wsdl:message>
7249
7250 <wsdl:message name="batchForward">
7251   <wsdl:part name="batchForward" element="batchForward" />
7252 </wsdl:message>
7253 <wsdl:message name="batchForwardResponse">
7254   <wsdl:part name="batchForwardResponse" element="batchForwardResponse" />
7255 </wsdl:message>
7256
7257 <wsdl:message name="getAttachment">
7258   <wsdl:part name="getAttachment" element="getAttachment" />
7259 </wsdl:message>
7260 <wsdl:message name="getAttachmentResponse">
7261   <wsdl:part name="getAttachmentResponse" element="getAttachmentResponse" />
7262 </wsdl:message>
7263
7264 <wsdl:message name="getAttachmentInfos">
7265   <wsdl:part name="getAttachmentInfos" element="getAttachmentInfos" />
7266 </wsdl:message>
7267 <wsdl:message name="getAttachmentInfosResponse">
7268   <wsdl:part name="getAttachmentInfosResponse"
7269 element="getAttachmentInfosResponse" />
7270 </wsdl:message>
7271
7272 <wsdl:message name="getComments">
7273   <wsdl:part name="getComments" element="getComments" />
7274 </wsdl:message>
7275 <wsdl:message name="getCommentsResponse">
7276   <wsdl:part name="getCommentsResponse" element="getCommentsResponse" />
7277 </wsdl:message>
7278
7279 <wsdl:message name="getFault">
7280   <wsdl:part name="getFault" element="getFault" />
7281 </wsdl:message>
7282 <wsdl:message name="getFaultResponse">
7283   <wsdl:part name="getFaultResponse" element="getFaultResponse" />
7284 </wsdl:message>
7285
7286 <wsdl:message name="getInput">
7287   <wsdl:part name="getInput" element="getInput" />
7288 </wsdl:message>
7289 <wsdl:message name="getInputResponse">
7290   <wsdl:part name="getInputResponse" element="getInputResponse" />
7291 </wsdl:message>
7292
7293 <wsdl:message name="getOutcome">
7294   <wsdl:part name="getOutcome" element="getOutcome" />
7295 </wsdl:message>
7296 <wsdl:message name="getOutcomeResponse">
7297   <wsdl:part name="getOutcomeResponse" element="getOutcomeResponse" />
7298 </wsdl:message>
7299
7300 <wsdl:message name="getOutput">
7301   <wsdl:part name="getOutput" element="getOutput" />
7302 </wsdl:message>

```



```

7303 <wsdl:message name="getOutputResponse">
7304 <wsdl:part name="getOutputResponse" element="getOutputResponse"/>
7305 </wsdl:message>
7306
7307 <wsdl:message name="getParentTask">
7308 <wsdl:part name="getParentTask" element="getParentTask"/>
7309 </wsdl:message>
7310 <wsdl:message name="getParentTaskResponse">
7311 <wsdl:part name="getParentTaskResponse" element="getParentTaskResponse"/>
7312 </wsdl:message>
7313
7314 <wsdl:message name="getParentTaskIdentifier">
7315 <wsdl:part name="getParentTaskIdentifier"
7316 element="getParentTaskIdentifier"/>
7317 </wsdl:message>
7318 <wsdl:message name="getParentTaskIdentifierResponse">
7319 <wsdl:part name="getParentTaskIdentifierResponse"
7320 element="getParentTaskIdentifierResponse"/>
7321 </wsdl:message>
7322
7323 <wsdl:message name="getRendering">
7324 <wsdl:part name="getRendering" element="getRendering"/>
7325 </wsdl:message>
7326 <wsdl:message name="getRenderingResponse">
7327 <wsdl:part name="getRenderingResponse" element="getRenderingResponse"/>
7328 </wsdl:message>
7329
7330 <wsdl:message name="getRenderingTypes">
7331 <wsdl:part name="getRenderingTypes" element="getRenderingTypes"/>
7332 </wsdl:message>
7333 <wsdl:message name="getRenderingTypesResponse">
7334 <wsdl:part name="getRenderingTypesResponse"
7335 element="getRenderingTypesResponse"/>
7336 </wsdl:message>
7337
7338 <wsdl:message name="getSubtaskIdentifiers">
7339 <wsdl:part name="getSubtaskIdentifiers" element="getSubtaskIdentifiers"/>
7340 </wsdl:message>
7341 <wsdl:message name="getSubtaskIdentifiersResponse">
7342 <wsdl:part name="getSubtaskIdentifiersResponse"
7343 element="getSubtaskIdentifiersResponse"/>
7344 </wsdl:message>
7345
7346 <wsdl:message name="getSubtasks">
7347 <wsdl:part name="getSubtasks" element="getSubtasks"/>
7348 </wsdl:message>
7349 <wsdl:message name="getSubtasksResponse">
7350 <wsdl:part name="getSubtasksResponse" element="getSubtasksResponse"/>
7351 </wsdl:message>
7352
7353 <wsdl:message name="getTaskDescription">
7354 <wsdl:part name="getTaskDescription" element="getTaskDescription"/>
7355 </wsdl:message>
7356 <wsdl:message name="getTaskDescriptionResponse">
7357 <wsdl:part name="getTaskDescriptionResponse"
7358 element="getTaskDescriptionResponse"/>
7359 </wsdl:message>
7360

```

```

7361 <wsdl:message name="getTaskDetails">
7362 <wsdl:part name="getTaskDetails" element="getTaskDetails"/>
7363 </wsdl:message>
7364 <wsdl:message name="getTaskDetailsResponse">
7365 <wsdl:part name="getTaskDetailsResponse"
7366 element="getTaskDetailsResponse"/>
7367 </wsdl:message>
7368
7369 <wsdl:message name="getTaskHistory">
7370 <wsdl:part name="getTaskHistory" element="getTaskHistory"/>
7371 </wsdl:message>
7372 <wsdl:message name="getTaskHistoryResponse">
7373 <wsdl:part name="getTaskHistoryResponse"
7374 element="getTaskHistoryResponse"/>
7375 </wsdl:message>
7376
7377 <wsdl:message name="getTaskInstanceData">
7378 <wsdl:part name="getTaskInstanceData" element="getTaskInstanceData"/>
7379 </wsdl:message>
7380 <wsdl:message name="getTaskInstanceDataResponse">
7381 <wsdl:part name="getTaskInstanceDataResponse"
7382 element="getTaskInstanceDataResponse"/>
7383 </wsdl:message>
7384
7385 <wsdl:message name="getTaskOperations">
7386 <wsdl:part name="getTaskOperations" element="getTaskOperations"/>
7387 </wsdl:message>
7388 <wsdl:message name="getTaskOperationsResponse">
7389 <wsdl:part name="getTaskOperationsResponse"
7390 element="getTaskOperationsResponse"/>
7391 </wsdl:message>
7392
7393 <wsdl:message name="hasSubtasks">
7394 <wsdl:part name="hasSubtasks" element="hasSubtasks"/>
7395 </wsdl:message>
7396 <wsdl:message name="hasSubtasksResponse">
7397 <wsdl:part name="hasSubtasksResponse" element="hasSubtasksResponse"/>
7398 </wsdl:message>
7399
7400 <wsdl:message name="instantiateSubtask">
7401 <wsdl:part name="instantiateSubtask" element="instantiateSubtask"/>
7402 </wsdl:message>
7403 <wsdl:message name="instantiateSubtaskResponse">
7404 <wsdl:part name="instantiateSubtaskResponse"
7405 element="instantiateSubtaskResponse"/>
7406 </wsdl:message>
7407
7408 <wsdl:message name="isSubtask">
7409 <wsdl:part name="isSubtask" element="isSubtask"/>
7410 </wsdl:message>
7411 <wsdl:message name="isSubtaskResponse">
7412 <wsdl:part name="isSubtaskResponse" element="isSubtaskResponse"/>
7413 </wsdl:message>
7414
7415 <wsdl:message name="release">
7416 <wsdl:part name="release" element="release"/>
7417 </wsdl:message>
7418 <wsdl:message name="releaseResponse">

```

```
7419     <wsdl:part name="releaseResponse" element="releaseResponse" />
7420 </wsdl:message>
7421
7422 <wsdl:message name="batchRelease">
7423     <wsdl:part name="batchRelease" element="batchRelease" />
7424 </wsdl:message>
7425 <wsdl:message name="batchReleaseResponse">
7426     <wsdl:part name="batchReleaseResponse" element="batchReleaseResponse" />
7427 </wsdl:message>
7428
7429 <wsdl:message name="remove">
7430     <wsdl:part name="remove" element="remove" />
7431 </wsdl:message>
7432 <wsdl:message name="removeResponse">
7433     <wsdl:part name="removeResponse" element="removeResponse" />
7434 </wsdl:message>
7435
7436 <wsdl:message name="batchRemove">
7437     <wsdl:part name="batchRemove" element="batchRemove" />
7438 </wsdl:message>
7439 <wsdl:message name="batchRemoveResponse">
7440     <wsdl:part name="batchRemoveResponse" element="batchRemoveResponse" />
7441 </wsdl:message>
7442
7443 <wsdl:message name="resume">
7444     <wsdl:part name="resume" element="resume" />
7445 </wsdl:message>
7446 <wsdl:message name="resumeResponse">
7447     <wsdl:part name="resumeResponse" element="resumeResponse" />
7448 </wsdl:message>
7449
7450 <wsdl:message name="batchResume">
7451     <wsdl:part name="batchResume" element="batchResume" />
7452 </wsdl:message>
7453 <wsdl:message name="batchResumeResponse">
7454     <wsdl:part name="batchResumeResponse" element="batchResumeResponse" />
7455 </wsdl:message>
7456
7457 <wsdl:message name="setFault">
7458     <wsdl:part name="setFault" element="setFault" />
7459 </wsdl:message>
7460 <wsdl:message name="setFaultResponse">
7461     <wsdl:part name="setFaultResponse" element="setFaultResponse" />
7462 </wsdl:message>
7463
7464 <wsdl:message name="setOutput">
7465     <wsdl:part name="setOutput" element="setOutput" />
7466 </wsdl:message>
7467 <wsdl:message name="setOutputResponse">
7468     <wsdl:part name="setOutputResponse" element="setOutputResponse" />
7469 </wsdl:message>
7470
7471 <wsdl:message name="setPriority">
7472     <wsdl:part name="setPriority" element="setPriority" />
7473 </wsdl:message>
7474 <wsdl:message name="setPriorityResponse">
7475     <wsdl:part name="setPriorityResponse" element="setPriorityResponse" />
7476 </wsdl:message>
```

```

7477 <wsdl:message name="batchSetPriority">
7478 <wsdl:part name="batchSetPriority" element="batchSetPriority"/>
7479 </wsdl:message>
7480 <wsdl:message name="batchSetPriorityResponse">
7481 <wsdl:part name="batchSetPriorityResponse"
7482 element="batchSetPriorityResponse"/>
7483 </wsdl:message>
7484
7485 <wsdl:message name="setTaskCompletionDeadlineExpression">
7486 <wsdl:part name="setTaskCompletionDeadlineExpression"
7487 element="setTaskCompletionDeadlineExpression"/>
7488 </wsdl:message>
7489 <wsdl:message name="setTaskCompletionDeadlineExpressionResponse">
7490 <wsdl:part name="setTaskCompletionDeadlineExpressionResponse"
7491 element="setTaskCompletionDeadlineExpressionResponse"/>
7492 </wsdl:message>
7493
7494 <wsdl:message name="setTaskCompletionDurationExpression">
7495 <wsdl:part name="setTaskCompletionDurationExpression"
7496 element="setTaskCompletionDurationExpression"/>
7497 </wsdl:message>
7498 <wsdl:message name="setTaskCompletionDurationExpressionResponse">
7499 <wsdl:part name="setTaskCompletionDurationExpressionResponse"
7500 element="setTaskCompletionDurationExpressionResponse"/>
7501 </wsdl:message>
7502
7503 <wsdl:message name="setTaskStartDeadlineExpression">
7504 <wsdl:part name="setTaskStartDeadlineExpression"
7505 element="setTaskStartDeadlineExpression"/>
7506 </wsdl:message>
7507 <wsdl:message name="setTaskStartDeadlineExpressionResponse">
7508 <wsdl:part name="setTaskStartDeadlineExpressionResponse"
7509 element="setTaskStartDeadlineExpressionResponse"/>
7510 </wsdl:message>
7511
7512 <wsdl:message name="setTaskStartDurationExpression">
7513 <wsdl:part name="setTaskStartDurationExpression"
7514 element="setTaskStartDurationExpression"/>
7515 </wsdl:message>
7516 <wsdl:message name="setTaskStartDurationExpressionResponse">
7517 <wsdl:part name="setTaskStartDurationExpressionResponse"
7518 element="setTaskStartDurationExpressionResponse"/>
7519 </wsdl:message>
7520
7521 <wsdl:message name="skip">
7522 <wsdl:part name="skip" element="skip"/>
7523 </wsdl:message>
7524 <wsdl:message name="skipResponse">
7525 <wsdl:part name="skipResponse" element="skipResponse"/>
7526 </wsdl:message>
7527
7528 <wsdl:message name="batchSkip">
7529 <wsdl:part name="batchSkip" element="batchSkip"/>
7530 </wsdl:message>
7531 <wsdl:message name="batchSkipResponse">
7532 <wsdl:part name="batchSkipResponse" element="batchSkipResponse"/>
7533 </wsdl:message>
7534

```

```

7535 <wsdl:message name="start">
7536   <wsdl:part name="start" element="start"/>
7537 </wsdl:message>
7538 <wsdl:message name="startResponse">
7539   <wsdl:part name="startResponse" element="startResponse"/>
7540 </wsdl:message>
7541
7542 <wsdl:message name="batchStart">
7543   <wsdl:part name="batchStart" element="batchStart"/>
7544 </wsdl:message>
7545 <wsdl:message name="batchStartResponse">
7546   <wsdl:part name="batchStartResponse" element="batchStartResponse"/>
7547 </wsdl:message>
7548
7549 <wsdl:message name="stop">
7550   <wsdl:part name="stop" element="stop"/>
7551 </wsdl:message>
7552 <wsdl:message name="stopResponse">
7553   <wsdl:part name="stopResponse" element="stopResponse"/>
7554 </wsdl:message>
7555
7556 <wsdl:message name="batchStop">
7557   <wsdl:part name="batchStop" element="batchStop"/>
7558 </wsdl:message>
7559 <wsdl:message name="batchStopResponse">
7560   <wsdl:part name="batchStopResponse" element="batchStopResponse"/>
7561 </wsdl:message>
7562
7563 <wsdl:message name="release">
7564   <wsdl:part name="release" element="release"/>
7565 </wsdl:message>
7566 <wsdl:message name="releaseResponse">
7567   <wsdl:part name="releaseResponse" element="releaseResponse"/>
7568 </wsdl:message>
7569
7570 <wsdl:message name="batchRelease">
7571   <wsdl:part name="batchRelease" element="batchRelease"/>
7572 </wsdl:message>
7573 <wsdl:message name="batchReleaseResponse">
7574   <wsdl:part name="batchReleaseResponse" element="batchReleaseResponse"/>
7575 </wsdl:message>
7576
7577 <wsdl:message name="suspend">
7578   <wsdl:part name="suspend" element="suspend"/>
7579 </wsdl:message>
7580 <wsdl:message name="suspendResponse">
7581   <wsdl:part name="suspendResponse" element="suspendResponse"/>
7582 </wsdl:message>
7583
7584 <wsdl:message name="batchSuspend">
7585   <wsdl:part name="batchSuspend" element="batchSuspend"/>
7586 </wsdl:message>
7587 <wsdl:message name="batchSuspendResponse">
7588   <wsdl:part name="batchSuspendResponse" element="batchSuspendResponse"/>
7589 </wsdl:message>
7590
7591 <wsdl:message name="suspendUntil">
7592

```

```

7593     <wsdl:part name="suspendUntil" element="suspendUntil"/>
7594 </wsdl:message>
7595 <wsdl:message name="suspendUntilResponse">
7596     <wsdl:part name="suspendUntilResponse" element="suspendUntilResponse"/>
7597 </wsdl:message>
7598
7599 <wsdl:message name="batchSuspendUntil">
7600     <wsdl:part name="batchSuspendUntil" element="batchSuspendUntil"/>
7601 </wsdl:message>
7602 <wsdl:message name="batchSuspendUntilResponse">
7603     <wsdl:part name="batchSuspendUntilResponse"
7604 element="batchSuspendUntilResponse"/>
7605 </wsdl:message>
7606
7607 <wsdl:message name="resumeupdateComment">
7608 <wsdl:part name="resume" element="resume"/>
7609 </wsdl:message>
7610 <wsdl:message name="resumeResponse">
7611 <wsdl:part name="resumeResponse" element="resumeResponse"/>
7612 </wsdl:message>
7613
7614 <wsdl:message name="batchResume">
7615 <wsdl:part name="batchResume" element="batchResume"/>
7616 </wsdl:message>
7617 <wsdl:message name="batchResumeResponse">
7618 <wsdl:part name="batchResumeResponse" element="batchResumeResponse"/>
7619 </wsdl:message>
7620
7621 <wsdl:message name="complete">
7622 <wsdl:part name="complete" element="complete"/>
7623 </wsdl:message>
7624 <wsdl:message name="completeResponse">
7625 <wsdl:part name="completeResponse" element="completeResponse"/>
7626 </wsdl:message>
7627
7628 <wsdl:message name="batchComplete">
7629 <wsdl:part name="batchComplete" element="batchComplete"/>
7630 </wsdl:message>
7631 <wsdl:message name="batchCompleteResponse">
7632 <wsdl:part name="batchCompleteResponse" element="batchCompleteResponse"/>
7633 </wsdl:message>
7634
7635 <wsdl:message name="remove">
7636 <wsdl:part name="remove" element="remove"/>
7637 </wsdl:message>
7638 <wsdl:message name="removeResponse">
7639 <wsdl:part name="removeResponse" element="removeResponse"/>
7640 </wsdl:message>
7641
7642 <wsdl:message name="batchRemove">
7643 <wsdl:part name="batchRemove" element="batchRemove"/>
7644 </wsdl:message>
7645 <wsdl:message name="batchRemoveResponse">
7646 <wsdl:part name="batchRemoveResponse" element="batchRemoveResponse"/>
7647 </wsdl:message>
7648
7649 <wsdl:message name="fail">
7650 <wsdl:part name="fail" element="fail"/>

```

```

7651 </wsdl:message>
7652 <wsdl:message name="failResponse">
7653 <wsdl:part name="failResponse" element="failResponse"/>
7654 </wsdl:message>
7655
7656 <wsdl:message name="batchFail">
7657 <wsdl:part name="batchFail" element="batchFail"/>
7658 </wsdl:message>
7659 <wsdl:message name="batchFailResponse">
7660 <wsdl:part name="batchFailResponse" element="batchFailResponse"/>
7661 </wsdl:message>
7662
7663 <wsdl:message name="setPriority">
7664 <wsdl:part name="setPriority" element="setPriority"/>
7665 </wsdl:message>
7666 <wsdl:message name="setPriorityResponse">
7667 <wsdl:part name="setPriorityResponse" element="setPriorityResponse"/>
7668 </wsdl:message>
7669
7670 <wsdl:message name="batchSetPriority">
7671 <wsdl:part name="batchSetPriority" element="batchSetPriority"/>
7672 </wsdl:message>
7673 <wsdl:message name="batchSetPriorityResponse">
7674 <wsdl:part name="batchSetPriorityResponse"
7675 element="batchSetPriorityResponse"/>
7676 </wsdl:message>
7677
7678 <wsdl:part name="updateComment" element="updateComment" />
7679 </wsdl:message>
7680 <wsdl:message name="addAttachment">
7681 <wsdl:part name="addAttachment" element="addAttachment"/>
7682 </wsdl:message>
7683 <wsdl:message name="addAttachmentResponse">
7684 <wsdl:part name="addAttachmentResponse" element="addAttachmentResponse"/>
7685 </wsdl:message>
7686
7687 <wsdl:message name="getAttachmentInfosupdateCommentResponse">
7688 <wsdl:part name="getAttachmentInfosupdateCommentResponse"
7689 element="getAttachmentInfosupdateCommentResponse" />
7690 </wsdl:message>
7691 <wsdl:message name="getAttachmentInfosResponse">
7692 <wsdl:part name="getAttachmentInfosResponse"
7693 element="getAttachmentInfosResponse"/>
7694 </wsdl:message>
7695
7696 <wsdl:message name="getAttachment">
7697 <wsdl:part name="getAttachment" element="getAttachment"/>
7698 </wsdl:message>
7699 <wsdl:message name="getAttachmentResponse">
7700 <wsdl:part name="getAttachmentResponse" element="getAttachmentResponse"/>
7701 </wsdl:message>
7702
7703 <wsdl:message name="deleteAttachment">
7704 <wsdl:part name="deleteAttachment" element="deleteAttachment"/>
7705 </wsdl:message>
7706 <wsdl:message name="deleteAttachmentResponse">
7707 <wsdl:part name="deleteAttachmentResponse"
7708 element="deleteAttachmentResponse"/>

```

```
7709 </wsdl:message>
7710
7711 <wsdl:message name="addComment">
7712 <wsdl:part name="addComment" element="addComment"/>
7713 </wsdl:message>
7714 <wsdl:message name="addCommentResponse">
7715 <wsdl:part name="addCommentResponse" element="addCommentResponse"/>
7716 </wsdl:message>
7717
7718 <wsdl:message name="getComments">
7719 <wsdl:part name="getComments" element="getComments"/>
7720 </wsdl:message>
7721 <wsdl:message name="getCommentsResponse">
7722 <wsdl:part name="getCommentsResponse" element="getCommentsResponse"/>
7723 </wsdl:message>
7724
7725 <wsdl:message name="skip">
7726 <wsdl:part name="skip" element="skip"/>
7727 </wsdl:message>
7728 <wsdl:message name="skipResponse">
7729 <wsdl:part name="skipResponse" element="skipResponse"/>
7730 </wsdl:message>
7731
7732 <wsdl:message name="batchSkip">
7733 <wsdl:part name="batchSkip" element="batchSkip"/>
7734 </wsdl:message>
7735 <wsdl:message name="batchSkipResponse">
7736 <wsdl:part name="batchSkipResponse" element="batchSkipResponse"/>
7737 </wsdl:message>
7738
7739 <wsdl:message name="forward">
7740 <wsdl:part name="forward" element="forward"/>
7741 </wsdl:message>
7742 <wsdl:message name="forwardResponse">
7743 <wsdl:part name="forwardResponse" element="forwardResponse"/>
7744 </wsdl:message>
7745
7746 <wsdl:message name="batchForward">
7747 <wsdl:part name="batchForward" element="batchForward"/>
7748 </wsdl:message>
7749 <wsdl:message name="batchForwardResponse">
7750 <wsdl:part name="batchForwardResponse" element="batchForwardResponse"/>
7751 </wsdl:message>
7752
7753 <wsdl:message name="delegate">
7754 <wsdl:part name="delegate" element="delegate"/>
7755 </wsdl:message>
7756 <wsdl:message name="delegateResponse">
7757 <wsdl:part name="delegateResponse" element="delegateResponse"/>
7758 </wsdl:message>
7759
7760 <wsdl:message name="batchDelegate">
7761 <wsdl:part name="batchDelegate" element="batchDelegate"/>
7762 </wsdl:message>
7763 <wsdl:message name="batchDelegateResponse">
7764 <wsdl:part name="batchDelegateResponse" element="batchDelegateResponse"/>
7765 </wsdl:message>
7766
```



```
7767 <wsdl:message name="getRendering">
7768 <wsdl:part name="getRendering" element="getRendering"/>
7769 </wsdl:message>
7770 <wsdl:message name="getRenderingResponse">
7771 <wsdl:part name="getRenderingResponse" element="getRenderingResponse"/>
7772 </wsdl:message>
7773
7774 <wsdl:message name="getRenderingTypes">
7775 <wsdl:part name="getRenderingTypes" element="getRenderingTypes"/>
7776 </wsdl:message>
7777 <wsdl:message name="getRenderingTypesResponse">
7778 <wsdl:part name="getRenderingTypesResponse"
7779 element="getRenderingTypesResponse"/>
7780 </wsdl:message>
7781
7782 <wsdl:message name="getTaskDetails">
7783 <wsdl:part name="getTaskDetails" element="getTaskDetails"/>
7784 </wsdl:message>
7785 <wsdl:message name="getTaskDetailsResponse">
7786 <wsdl:part name="getTaskDetailsResponse"
7787 element="getTaskDetailsResponse"/>
7788 </wsdl:message>
7789
7790 <wsdl:message name="getTaskDescription">
7791 <wsdl:part name="getTaskDescription" element="getTaskDescription"/>
7792 </wsdl:message>
7793 <wsdl:message name="getTaskDescriptionResponse">
7794 <wsdl:part name="getTaskDescriptionResponse"
7795 element="getTaskDescriptionResponse"/>
7796 </wsdl:message>
7797
7798 <wsdl:message name="setOutput">
7799 <wsdl:part name="setOutput" element="setOutput"/>
7800 </wsdl:message>
7801 <wsdl:message name="setOutputResponse">
7802 <wsdl:part name="setOutputResponse" element="setOutputResponse"/>
7803 </wsdl:message>
7804
7805 <wsdl:message name="deleteOutput">
7806 <wsdl:part name="deleteOutput" element="deleteOutput"/>
7807 </wsdl:message>
7808 <wsdl:message name="deleteOutputResponse">
7809 <wsdl:part name="deleteOutputResponse" element="deleteOutputResponse"/>
7810 </wsdl:message>
7811
7812 <wsdl:message name="setFault">
7813 <wsdl:part name="setFault" element="setFault"/>
7814 </wsdl:message>
7815 <wsdl:message name="setFaultResponse">
7816 <wsdl:part name="setFaultResponse" element="setFaultResponse"/>
7817 </wsdl:message>
7818
7819 <wsdl:message name="deleteFault">
7820 <wsdl:part name="deleteFault" element="deleteFault"/>
7821 </wsdl:message>
7822 <wsdl:message name="deleteFaultResponse">
7823 <wsdl:part name="deleteFaultResponse" element="deleteFaultResponse"/>
7824 </wsdl:message>
```

```

7825 <wsdl:message name="getInput">
7826 <wsdl:part name="getInput" element="getInput"/>
7827 </wsdl:message>
7828 <wsdl:message name="getInputResponse">
7829 <wsdl:part name="getInputResponse" element="getInputResponse"/>
7830 </wsdl:message>
7831
7832 <wsdl:message name="getOutput">
7833 <wsdl:part name="getOutput" element="getOutput"/>
7834 </wsdl:message>
7835 <wsdl:message name="getOutputResponse">
7836 <wsdl:part name="getOutputResponse" element="getOutputResponse"/>
7837 </wsdl:message>
7838
7839 <wsdl:message name="getFault">
7840 <wsdl:part name="getFault" element="getFault"/>
7841 </wsdl:message>
7842 <wsdl:message name="getFaultResponse">
7843 <wsdl:part name="getFaultResponse" element="getFaultResponse"/>
7844 </wsdl:message>
7845
7846
7847 <wsdl:message name="getMyTaskAbstracts">
7848   <wsdl:part name="getMyTaskAbstracts" element="getMyTaskAbstracts"/>
7849 </wsdl:message>
7850 <wsdl:message name="getMyTaskAbstractsResponse">
7851   <wsdl:part name="getMyTaskAbstractsResponse"
7852 element="getMyTaskAbstractsResponse" />
7853 </wsdl:message>
7854
7855 <wsdl:message name="getMyTaskDetails">
7856   <wsdl:part name="getMyTaskDetails" element="getMyTaskDetails"/>
7857 </wsdl:message>
7858 <wsdl:message name="getMyTaskDetailsResponse">
7859   <wsdl:part name="getMyTaskDetailsResponse"
7860 element="getMyTaskDetailsResponse" />
7861 </wsdl:message>
7862
7863 <wsdl:message name="query">
7864   <wsdl:part name="query" element="query"/>
7865 </wsdl:message>
7866 <wsdl:message name="queryResponse">
7867   <wsdl:part name="queryResponse" element="queryResponse"/>
7868 </wsdl:message>
7869
7870 <wsdl:message name="activate">
7871   <wsdl:part name="activate" element="activate"/>
7872 </wsdl:message>
7873 <wsdl:message name="activateResponse">
7874   <wsdl:part name="activateResponse" element="activateResponse"/>
7875 </wsdl:message>
7876
7877 <wsdl:message name="batchActivate">
7878   <wsdl:part name="batchActivate" element="batchActivate"/>
7879 </wsdl:message>
7880 <wsdl:message name="batchActivateResponse">
7881   <wsdl:part name="batchActivateResponse" element="batchActivateResponse"/>
7882 </wsdl:message>

```

```

7883
7884 <wsdl:message name="nominate">
7885   <wsdl:part name="nominate" element="nominate"/>
7886 </wsdl:message>
7887 <wsdl:message name="nominateResponse">
7888   <wsdl:part name="nominateResponse" element="nominateResponse"/>
7889 </wsdl:message>
7890
7891 <wsdl:message name="batchNominate">
7892   <wsdl:part name="batchNominate" element="batchNominate"/>
7893 </wsdl:message>
7894 <wsdl:message name="batchNominateResponse">
7895   <wsdl:part name="batchNominateResponse" element="batchNominateResponse"/>
7896 </wsdl:message>
7897
7898 <wsdl:message name="setGenericHumanRole">
7899   <wsdl:part name="setGenericHumanRole" element="setGenericHumanRole"/>
7900 </wsdl:message>
7901 <wsdl:message name="setGenericHumanRoleResponse">
7902   <wsdl:part name="setGenericHumanRoleResponse"
7903 element="setGenericHumanRoleResponse"/>
7904 </wsdl:message>
7905
7906 <wsdl:message name="batchSetGenericHumanRole">
7907   <wsdl:part name="batchSetGenericHumanRole"
7908 element="batchSetGenericHumanRole"/>
7909 </wsdl:message>
7910 <wsdl:message name="batchSetGenericHumanRoleResponse">
7911   <wsdl:part name="batchSetGenericHumanRoleResponse"
7912 element="batchSetGenericHumanRoleResponse"/>
7913 </wsdl:message>
7914
7915 <wsdl:message name="getOutcome">
7916 <wsdl:part name="getOutcome" element="getOutcome"/>
7917 </wsdl:message>
7918 <wsdl:message name="getOutcomeResponse">
7919 <wsdl:part name="getOutcomeResponse" element="getOutcomeResponse"/>
7920 </wsdl:message>
7921
7922 <wsdl:message name="getTaskOperations">
7923 <wsdl:part name="getTaskOperations" element="getTaskOperations"/>
7924 </wsdl:message>
7925 <wsdl:message name="getTaskOperationsResponse">
7926 <wsdl:part name="getTaskOperationsResponse"
7927 element="getTaskOperationsResponse"/>
7928 </wsdl:message>
7929
7930 <wsdl:message name="getTaskInstanceData">
7931 <wsdl:part name="getTaskInstanceData" element="getTaskInstanceData"/>
7932 </wsdl:message>
7933 <wsdl:message name="getTaskInstanceDataResponse">
7934 <wsdl:part name="getTaskInstanceDataResponse"
7935 element="getTaskInstanceDataResponse"/>
7936 </wsdl:message>
7937
7938 <wsdl:message name="getTaskHistory">
7939 <wsdl:part name="getTaskHistory" element="getTaskHistory"/>
7940 </wsdl:message>

```

```

7941 <wsdl:message name="getTaskHistoryResponse">
7942 <wsdl:part name="getTaskHistoryResponse"
7943 element="getTaskHistoryResponse"/>
7944 </wsdl:message>
7945
7946 <wsdl:message name="setTaskStartDeadlineExpression">
7947 <wsdl:part name="setTaskStartDeadlineExpression"
7948 element="setTaskStartDeadlineExpression"/>
7949 </wsdl:message>
7950 <wsdl:message name="setTaskStartDeadlineExpressionResponse">
7951 <wsdl:part name="setTaskStartDeadlineExpressionResponse"
7952 element="setTaskStartDeadlineExpressionResponse"/>
7953 </wsdl:message>
7954
7955 <wsdl:message name="setTaskStartDurationExpression">
7956 <wsdl:part name="setTaskStartDurationExpression"
7957 element="setTaskStartDurationExpression"/>
7958 </wsdl:message>
7959 <wsdl:message name="setTaskStartDurationExpressionResponse">
7960 <wsdl:part name="setTaskStartDurationExpressionResponse"
7961 element="setTaskStartDurationExpressionResponse"/>
7962 </wsdl:message>
7963
7964 <wsdl:message name="setTaskCompletionDeadlineExpression">
7965 <wsdl:part name="setTaskCompletionDeadlineExpression"
7966 element="setTaskCompletionDeadlineExpression"/>
7967 </wsdl:message>
7968 <wsdl:message name="setTaskCompletionDeadlineExpressionResponse">
7969 <wsdl:part name="setTaskCompletionDeadlineExpressionResponse"
7970 element="setTaskCompletionDeadlineExpressionResponse"/>
7971 </wsdl:message>
7972
7973 <wsdl:message name="setTaskCompletionDurationExpression">
7974 <wsdl:part name="setTaskCompletionDurationExpression"
7975 element="setTaskCompletionDurationExpression"/>
7976 </wsdl:message>
7977 <wsdl:message name="setTaskCompletionDurationExpressionResponse">
7978 <wsdl:part name="setTaskCompletionDurationExpressionResponse"
7979 element="setTaskCompletionDurationExpressionResponse"/>
7980 </wsdl:message>
7981
7982 <!-- Declaration of fault messages -->
7983 <wsdl:message name="illegalStateFault">
7984 <wsdl:part name="illegalState" element="illegalState"/>
7985 </wsdl:message>
7986 <wsdl:message name="illegalArgumentFault">
7987 <wsdl:part name="illegalArgument" element="illegalArgument"/>
7988 </wsdl:message>
7989 <wsdl:message name="illegalAccessFault">
7990 <wsdl:part name="illegalAccess" element="illegalAccess"/>
7991 </wsdl:message>
7992 <wsdl:message name="illegalOperationFault">
7993 <wsdl:part name="illegalOperation" element="illegalOperation"/>
7994 </wsdl:message>
7995 <wsdl:message name="recipientNotAllowed">
7996 <wsdl:part name="recipientNotAllowed" element="recipientNotAllowed"/>
7997 </wsdl:message>
7998

```

```

7999 <!-- Port type definition -->
8000 <wsdl:portType name="taskOperations">
8001
8002 <wsdl:operation name="addAttachment">
8003 <wsdl:input message="addAttachment" />
8004 <wsdl:output message="addAttachmentResponse" />
8005 <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
8006 <wsdl:fault name="illegalArgumentFault"
8007 message="illegalArgumentFault" />
8008 <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8009 <wsdl:fault name="illegalOperationFault"
8010 message="illegalOperationFault" />
8011 </wsdl:operation>
8012
8013 <wsdl:operation name="addComment">
8014 <wsdl:input message="addComment" />
8015 <wsdl:output message="addCommentResponse" />
8016 <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
8017 <wsdl:fault name="illegalArgumentFault"
8018 message="illegalArgumentFault" />
8019 <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8020 <wsdl:fault name="illegalOperationFault"
8021 message="illegalOperationFault" />
8022 </wsdl:operation>
8023
8024 <wsdl:operation name="claim">
8025 <wsdl:input message="claim" />
8026 <wsdl:output message="claimResponse" />
8027 <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
8028 <wsdl:fault name="illegalArgumentFault"
8029 message="illegalArgumentFault" />
8030 <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8031 <wsdl:fault name="illegalOperationFault"
8032 message="illegalOperationFault" />
8033 </wsdl:operation>
8034
8035 <wsdl:operation name="batchClaim">
8036 <wsdl:input message="batchClaim" />
8037 <wsdl:output message="batchClaimResponse" />
8038 </wsdl:operation>
8039
8040 <wsdl:operation name="complete">
8041 <wsdl:input message="complete" />
8042 <wsdl:output message="completeResponse" />
8043 <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
8044 <wsdl:fault name="illegalArgumentFault"
8045 message="illegalArgumentFault" />
8046 <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8047 <wsdl:fault name="illegalOperationFault"
8048 message="illegalOperationFault" />
8049 </wsdl:operation>
8050
8051 <wsdl:operation name="batchComplete">
8052 <wsdl:input message="batchComplete" />
8053 <wsdl:output message="batchCompleteResponse" />
8054 </wsdl:operation>
8055
8056 <wsdl:operation name="delegate">

```

```

8057     <wsdl:input message="delegate" />
8058     <wsdl:output message="delegateResponse" />
8059     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
8060     <wsdl:fault name="illegalArgumentFault"
8061 message="illegalArgumentFault" />
8062     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8063     <wsdl:fault name="illegalOperationFault"
8064 message="illegalOperationFault" />
8065     <wsdl:fault name="recipientNotAllowed" message="recipientNotAllowed" />
8066 </wsdl:operation>
8067
8068 <wsdl:operation name="batchDelegate">
8069   <wsdl:input message="batchDelegate" />
8070   <wsdl:output message="batchDelegateResponse" />
8071 </wsdl:operation>
8072
8073 <wsdl:operation name="deleteAttachment">
8074   <wsdl:input message="deleteAttachment" />
8075   <wsdl:output message="deleteAttachmentResponse" />
8076   <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
8077   <wsdl:fault name="illegalArgumentFault"
8078 message="illegalArgumentFault" />
8079   <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8080   <wsdl:fault name="illegalOperationFault"
8081 message="illegalOperationFault" />
8082 </wsdl:operation>
8083
8084 <wsdl:operation name="deleteComment">
8085   <wsdl:input message="deleteComment" />
8086   <wsdl:output message="deleteCommentResponse" />
8087   <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
8088   <wsdl:fault name="illegalArgumentFault"
8089 message="illegalArgumentFault" />
8090   <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8091   <wsdl:fault name="illegalOperationFault"
8092 message="illegalOperationFault" />
8093 </wsdl:operation>
8094
8095 <wsdl:operation name="deleteFault">
8096   <wsdl:input message="deleteFault" />
8097   <wsdl:output message="deleteFaultResponse" />
8098   <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
8099   <wsdl:fault name="illegalArgumentFault"
8100 message="illegalArgumentFault" />
8101   <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8102   <wsdl:fault name="illegalOperationFault"
8103 message="illegalOperationFault" />
8104 </wsdl:operation>
8105
8106 <wsdl:operation name="deleteOutput">
8107   <wsdl:input message="deleteOutput" />
8108   <wsdl:output message="deleteOutputResponse" />
8109   <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
8110   <wsdl:fault name="illegalArgumentFault"
8111 message="illegalArgumentFault" />
8112   <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8113   <wsdl:fault name="illegalOperationFault"
8114 message="illegalOperationFault" />

```

```

8115 </wsdl:operation>
8116
8117 <wsdl:operation name="fail">
8118 <wsdl:input message="fail"/>
8119 <wsdl:output message="failResponse"/>
8120 <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
8121 <wsdl:fault name="illegalArgumentFault"
8122 message="illegalArgumentFault"/>
8123 <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
8124 <wsdl:fault name="illegalOperationFault"
8125 message="illegalOperationFault"/>
8126 </wsdl:operation>
8127
8128 <wsdl:operation name="batchFail">
8129 <wsdl:input message="batchFail"/>
8130 <wsdl:output message="batchFailResponse"/>
8131 </wsdl:operation>
8132
8133 <wsdl:operation name="forward">
8134 <wsdl:input message="forward"/>
8135 <wsdl:output message="forwardResponse"/>
8136 <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
8137 <wsdl:fault name="illegalArgumentFault"
8138 message="illegalArgumentFault"/>
8139 <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
8140 <wsdl:fault name="illegalOperationFault"
8141 message="illegalOperationFault"/>
8142 </wsdl:operation>
8143
8144 <wsdl:operation name="batchForward">
8145 <wsdl:input message="batchForward"/>
8146 <wsdl:output message="batchForwardResponse"/>
8147 </wsdl:operation>
8148
8149 <wsdl:operation name="getAttachment">
8150 <wsdl:input message="getAttachment"/>
8151 <wsdl:output message="getAttachmentResponse"/>
8152 <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
8153 <wsdl:fault name="illegalArgumentFault"
8154 message="illegalArgumentFault"/>
8155 <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
8156 <wsdl:fault name="illegalOperationFault"
8157 message="illegalOperationFault"/>
8158 </wsdl:operation>
8159
8160 <wsdl:operation name="getAttachmentInfos">
8161 <wsdl:input message="getAttachmentInfos"/>
8162 <wsdl:output message="getAttachmentInfosResponse"/>
8163 <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
8164 <wsdl:fault name="illegalArgumentFault"
8165 message="illegalArgumentFault"/>
8166 <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
8167 <wsdl:fault name="illegalOperationFault"
8168 message="illegalOperationFault"/>
8169 </wsdl:operation>
8170
8171 <wsdl:operation name="getComments">
8172 <wsdl:input message="getComments"/>

```



```

8173     <wsdl:output message="getCommentsResponse" />
8174     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
8175     <wsdl:fault name="illegalArgumentFault"
8176 message="illegalArgumentFault" />
8177     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8178     <wsdl:fault name="illegalOperationFault"
8179 message="illegalOperationFault" />
8180   </wsdl:operation>
8181
8182   <wsdl:operation name="getFault">
8183     <wsdl:input message="getFault" />
8184     <wsdl:output message="getFaultResponse" />
8185     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
8186     <wsdl:fault name="illegalArgumentFault"
8187 message="illegalArgumentFault" />
8188     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8189     <wsdl:fault name="illegalOperationFault"
8190 message="illegalOperationFault" />
8191   </wsdl:operation>
8192
8193   <wsdl:operation name="getInput">
8194     <wsdl:input message="getInput" />
8195     <wsdl:output message="getInputResponse" />
8196     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
8197     <wsdl:fault name="illegalArgumentFault"
8198 message="illegalArgumentFault" />
8199     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8200     <wsdl:fault name="illegalOperationFault"
8201 message="illegalOperationFault" />
8202   </wsdl:operation>
8203
8204   <wsdl:operation name="getOutcome">
8205     <wsdl:input message="getOutcome" />
8206     <wsdl:output message="getOutcomeResponse" />
8207     <wsdl:fault name="illegalArgumentFault"
8208 message="illegalArgumentFault" />
8209     <wsdl:fault name="illegalOperationFault"
8210 message="illegalOperationFault" />
8211   </wsdl:operation>
8212
8213   <wsdl:operation name="getOutput">
8214     <wsdl:input message="getOutput" />
8215     <wsdl:output message="getOutputResponse" />
8216     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
8217     <wsdl:fault name="illegalArgumentFault"
8218 message="illegalArgumentFault" />
8219     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8220     <wsdl:fault name="illegalOperationFault"
8221 message="illegalOperationFault" />
8222   </wsdl:operation>
8223
8224   <wsdl:operation name="getParentTask">
8225     <wsdl:input message="getParentTask" />
8226     <wsdl:output message="getParentTaskResponse" />
8227     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
8228     <wsdl:fault name="illegalArgumentFault"
8229 message="illegalArgumentFault" />
8230     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />

```



```

8231     <wsdl:fault name="illegalOperationFault"
8232 message="illegalOperationFault" />
8233   </wsdl:operation>
8234
8235   <wsdl:operation name="getParentTaskIdentifier">
8236     <wsdl:input message="getParentTaskIdentifier" />
8237     <wsdl:output message="getParentTaskIdentifierResponse" />
8238     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
8239     <wsdl:fault name="illegalArgumentFault"
8240 message="illegalArgumentFault" />
8241     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8242     <wsdl:fault name="illegalOperationFault"
8243 message="illegalOperationFault" />
8244   </wsdl:operation>
8245
8246   <wsdl:operation name="getRendering">
8247     <wsdl:input message="getRendering" />
8248     <wsdl:output message="getRenderingResponse" />
8249     <wsdl:fault name="illegalArgumentFault"
8250 message="illegalArgumentFault" />
8251   </wsdl:operation>
8252
8253   <wsdl:operation name="getRenderingTypes">
8254     <wsdl:input message="getRenderingTypes" />
8255     <wsdl:output message="getRenderingTypesResponse" />
8256     <wsdl:fault name="illegalArgumentFault"
8257 message="illegalArgumentFault" />
8258   </wsdl:operation>
8259
8260   <wsdl:operation name="getSubtaskIdentifiers">
8261     <wsdl:input message="getSubtaskIdentifiers" />
8262     <wsdl:output message="getSubtaskIdentifiersResponse" />
8263     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
8264     <wsdl:fault name="illegalArgumentFault"
8265 message="illegalArgumentFault" />
8266     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8267     <wsdl:fault name="illegalOperationFault"
8268 message="illegalOperationFault" />
8269   </wsdl:operation>
8270
8271   <wsdl:operation name="getSubtasks">
8272     <wsdl:input message="getSubtasks" />
8273     <wsdl:output message="getSubtasksResponse" />
8274     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
8275     <wsdl:fault name="illegalArgumentFault"
8276 message="illegalArgumentFault" />
8277     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8278     <wsdl:fault name="illegalOperationFault"
8279 message="illegalOperationFault" />
8280   </wsdl:operation>
8281
8282   <wsdl:operation name="getTaskDescription">
8283     <wsdl:input message="getTaskDescription" />
8284     <wsdl:output message="getTaskDescriptionResponse" />
8285     <wsdl:fault name="illegalArgumentFault"
8286 message="illegalArgumentFault" />
8287   </wsdl:operation>
8288

```

```

8289     <wsdl:operation name="getTaskDetails">
8290         <wsdl:input message="getTaskDetails" />
8291         <wsdl:output message="getTaskDetailsResponse" />
8292         <wsdl:fault name="illegalArgumentFault"
8293 message="illegalArgumentFault" />
8294     </wsdl:operation>
8295
8296     <wsdl:operation name="getTaskHistory">
8297         <wsdl:input message="getTaskHistory" />
8298         <wsdl:output message="getTaskHistoryResponse" />
8299         <wsdl:fault name="illegalArgumentFault"
8300 message="illegalArgumentFault" />
8301         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8302         <wsdl:fault name="illegalOperationFault"
8303 message="illegalOperationFault" />
8304     </wsdl:operation>
8305
8306     <wsdl:operation name="getTaskInstanceData">
8307         <wsdl:input message="getTaskInstanceData" />
8308         <wsdl:output message="getTaskInstanceDataResponse" />
8309         <wsdl:fault name="illegalArgumentFault"
8310 message="illegalArgumentFault" />
8311         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8312         <wsdl:fault name="illegalOperationFault"
8313 message="illegalOperationFault" />
8314     </wsdl:operation>
8315
8316     <wsdl:operation name="getTaskOperations">
8317         <wsdl:input message="getTaskOperations" />
8318         <wsdl:output message="getTaskOperationsResponse" />
8319         <wsdl:fault name="illegalArgumentFault"
8320 message="illegalArgumentFault" />
8321         <wsdl:fault name="illegalOperationFault"
8322 message="illegalOperationFault" />
8323     </wsdl:operation>
8324
8325     <wsdl:operation name="hasSubtasks">
8326         <wsdl:input message="hasSubtasks" />
8327         <wsdl:output message="hasSubtasksResponse" />
8328         <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
8329         <wsdl:fault name="illegalArgumentFault"
8330 message="illegalArgumentFault" />
8331         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8332         <wsdl:fault name="illegalOperationFault"
8333 message="illegalOperationFault" />
8334     </wsdl:operation>
8335
8336     <wsdl:operation name="instantiateSubtask">
8337         <wsdl:input message="instantiateSubtask" />
8338         <wsdl:output message="instantiateSubtaskResponse" />
8339         <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
8340         <wsdl:fault name="illegalArgumentFault"
8341 message="illegalArgumentFault" />
8342         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8343         <wsdl:fault name="illegalOperationFault"
8344 message="illegalOperationFault" />
8345     </wsdl:operation>
8346

```

```

8347     <wsdl:operation name="isSubtask">
8348         <wsdl:input message="isSubtask" />
8349         <wsdl:output message="isSubtaskResponse" />
8350         <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
8351         <wsdl:fault name="illegalArgumentFault"
8352 message="illegalArgumentFault" />
8353         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8354         <wsdl:fault name="illegalOperationFault"
8355 message="illegalOperationFault" />
8356     </wsdl:operation>
8357
8358     <wsdl:operation name="release">
8359         <wsdl:input message="release" />
8360         <wsdl:output message="releaseResponse" />
8361         <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
8362         <wsdl:fault name="illegalArgumentFault"
8363 message="illegalArgumentFault" />
8364         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8365         <wsdl:fault name="illegalOperationFault"
8366 message="illegalOperationFault" />
8367     </wsdl:operation>
8368
8369     <wsdl:operation name="batchRelease">
8370         <wsdl:input message="batchRelease" />
8371         <wsdl:output message="batchReleaseResponse" />
8372     </wsdl:operation>
8373
8374     <wsdl:operation name="remove">
8375         <wsdl:input message="remove" />
8376         <wsdl:output message="removeResponse" />
8377         <wsdl:fault name="illegalArgumentFault"
8378 message="illegalArgumentFault" />
8379         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8380         <wsdl:fault name="illegalOperationFault"
8381 message="illegalOperationFault" />
8382     </wsdl:operation>
8383
8384     <wsdl:operation name="batchRemove">
8385         <wsdl:input message="batchRemove" />
8386         <wsdl:output message="batchRemoveResponse" />
8387     </wsdl:operation>
8388
8389     <wsdl:operation name="resume">
8390         <wsdl:input message="resume" />
8391         <wsdl:output message="resumeResponse" />
8392         <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
8393         <wsdl:fault name="illegalArgumentFault"
8394 message="illegalArgumentFault" />
8395         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8396         <wsdl:fault name="illegalOperationFault"
8397 message="illegalOperationFault" />
8398     </wsdl:operation>
8399
8400     <wsdl:operation name="batchResume">
8401         <wsdl:input message="batchResume" />
8402         <wsdl:output message="batchResumeResponse" />
8403     </wsdl:operation>
8404

```

```

8405 <wsdl:operation name="setFault">
8406 <wsdl:input message="setFault"/>
8407 <wsdl:output message="setFaultResponse"/>
8408 <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
8409 <wsdl:fault name="illegalArgumentFault"
8410 message="illegalArgumentFault"/>
8411 <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
8412 <wsdl:fault name="illegalOperationFault"
8413 message="illegalOperationFault"/>
8414 </wsdl:operation>
8415
8416 <wsdl:operation name="setOutput">
8417 <wsdl:input message="setOutput"/>
8418 <wsdl:output message="setOutputResponse"/>
8419 <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
8420 <wsdl:fault name="illegalArgumentFault"
8421 message="illegalArgumentFault"/>
8422 <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
8423 <wsdl:fault name="illegalOperationFault"
8424 message="illegalOperationFault"/>
8425 </wsdl:operation>
8426
8427 <wsdl:operation name="setPriority">
8428 <wsdl:input message="setPriority"/>
8429 <wsdl:output message="setPriorityResponse"/>
8430 <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
8431 <wsdl:fault name="illegalArgumentFault"
8432 message="illegalArgumentFault"/>
8433 <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
8434 <wsdl:fault name="illegalOperationFault"
8435 message="illegalOperationFault"/>
8436 </wsdl:operation>
8437
8438 <wsdl:operation name="batchSetPriority">
8439 <wsdl:input message="batchSetPriority"/>
8440 <wsdl:output message="batchSetPriorityResponse"/>
8441 </wsdl:operation>
8442
8443 <wsdl:operation name="setTaskCompletionDeadlineExpression">
8444 <wsdl:input message="setTaskCompletionDeadlineExpression"/>
8445 <wsdl:output message="setTaskCompletionDeadlineExpressionResponse"/>
8446 <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
8447 <wsdl:fault name="illegalArgumentFault"
8448 message="illegalArgumentFault"/>
8449 <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
8450 <wsdl:fault name="illegalOperationFault"
8451 message="illegalOperationFault"/>
8452 </wsdl:operation>
8453
8454 <wsdl:operation name="setTaskCompletionDurationExpression">
8455 <wsdl:input message="setTaskCompletionDurationExpression"/>
8456 <wsdl:output message="setTaskCompletionDurationExpressionResponse"/>
8457 <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
8458 <wsdl:fault name="illegalArgumentFault"
8459 message="illegalArgumentFault"/>
8460 <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
8461 <wsdl:fault name="illegalOperationFault"
8462 message="illegalOperationFault"/>

```

```

8463     </wsdl:operation>
8464
8465     <wsdl:operation name="setTaskStartDeadlineExpression">
8466         <wsdl:input message="setTaskStartDeadlineExpression" />
8467         <wsdl:output message="setTaskStartDeadlineExpressionResponse" />
8468         <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
8469         <wsdl:fault name="illegalArgumentFault"
8470 message="illegalArgumentFault" />
8471         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8472         <wsdl:fault name="illegalOperationFault"
8473 message="illegalOperationFault" />
8474     </wsdl:operation>
8475
8476     <wsdl:operation name="setTaskStartDurationExpression">
8477         <wsdl:input message="setTaskStartDurationExpression" />
8478         <wsdl:output message="setTaskStartDurationExpressionResponse" />
8479         <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
8480         <wsdl:fault name="illegalArgumentFault"
8481 message="illegalArgumentFault" />
8482         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8483         <wsdl:fault name="illegalOperationFault"
8484 message="illegalOperationFault" />
8485     </wsdl:operation>
8486
8487     <wsdl:operation name="skip">
8488         <wsdl:input message="skip" />
8489         <wsdl:output message="skipResponse" />
8490         <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
8491         <wsdl:fault name="illegalArgumentFault"
8492 message="illegalArgumentFault" />
8493         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8494         <wsdl:fault name="illegalOperationFault"
8495 message="illegalOperationFault" />
8496     </wsdl:operation>
8497
8498     <wsdl:operation name="batchSkip">
8499         <wsdl:input message="batchSkip" />
8500         <wsdl:output message="batchSkipResponse" />
8501     </wsdl:operation>
8502
8503     <wsdl:operation name="start">
8504         <wsdl:input message="start" />
8505         <wsdl:output message="startResponse" />
8506         <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
8507         <wsdl:fault name="illegalArgumentFault"
8508 message="illegalArgumentFault" />
8509         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8510         <wsdl:fault name="illegalOperationFault"
8511 message="illegalOperationFault" />
8512     </wsdl:operation>
8513
8514     <wsdl:operation name="batchStart">
8515         <wsdl:input message="batchStart" />
8516         <wsdl:output message="batchStartResponse" />
8517     </wsdl:operation>
8518
8519     <wsdl:operation name="stop">
8520         <wsdl:input message="stop" />

```

```

8521     <wsdl:output message="stopResponse" />
8522     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
8523     <wsdl:fault name="illegalArgumentFault"
8524 message="illegalArgumentFault" />
8525     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8526     <wsdl:fault name="illegalOperationFault"
8527 message="illegalOperationFault" />
8528     </wsdl:operation>
8529
8530     <wsdl:operation name="batchStop">
8531     <wsdl:input message="batchStop" />
8532     <wsdl:output message="batchStopResponse" />
8533     </wsdl:operation>
8534
8535      <wsdl:operation name="release">
8536      <wsdl:input message="release" />
8537      <wsdl:output message="releaseResponse" />
8538      <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
8539      <wsdl:fault name="illegalArgumentFault"
8540 message="illegalArgumentFault" />
8541      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8542      <wsdl:fault name="illegalOperationFault"
8543 message="illegalOperationFault" />
8544      </wsdl:operation>
8545
8546      <wsdl:operation name="batchRelease">
8547      <wsdl:input message="batchRelease" />
8548      <wsdl:output message="batchReleaseResponse" />
8549      </wsdl:operation>
8550
8551     <wsdl:operation name="suspend">
8552     <wsdl:input message="suspend" />
8553     <wsdl:output message="suspendResponse" />
8554     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
8555     <wsdl:fault name="illegalArgumentFault"
8556 message="illegalArgumentFault" />
8557     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8558     <wsdl:fault name="illegalOperationFault"
8559 message="illegalOperationFault" />
8560     </wsdl:operation>
8561
8562     <wsdl:operation name="batchSuspend">
8563     <wsdl:input message="batchSuspend" />
8564     <wsdl:output message="batchSuspendResponse" />
8565     </wsdl:operation>
8566
8567     <wsdl:operation name="suspendUntil">
8568     <wsdl:input message="suspendUntil" />
8569     <wsdl:output message="suspendUntilResponse" />
8570     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
8571     <wsdl:fault name="illegalArgumentFault"
8572 message="illegalArgumentFault" />
8573     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8574     <wsdl:fault name="illegalOperationFault"
8575 message="illegalOperationFault" />
8576     </wsdl:operation>
8577
8578     <wsdl:operation name="batchSuspendUntil">

```



```

8579     <wsdl:input message="batchSuspendUntil" />
8580     <wsdl:output message="batchSuspendUntilResponse" />
8581 </wsdl:operation>
8582
8583 <wsdl:operation name="resume">
8584 <wsdl:input message="resume" />
8585 <wsdl:output message="resumeResponse" />
8586 <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
8587 <wsdl:fault name="illegalArgumentFault"
8588 message="illegalArgumentFault" />
8589 <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8590 <wsdl:fault name="illegalOperationFault"
8591 message="illegalOperationFault" />
8592 </wsdl:operation>
8593
8594 <wsdl:operation name="batchResume">
8595 <wsdl:input message="batchResume" />
8596 <wsdl:output message="batchResumeResponse" />
8597 </wsdl:operation>
8598
8599 <wsdl:operation name="complete">
8600 <wsdl:input message="complete" />
8601 <wsdl:output message="completeResponse" />
8602 <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
8603 <wsdl:fault name="illegalArgumentFault"
8604 message="illegalArgumentFault" />
8605 <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8606 <wsdl:fault name="illegalOperationFault"
8607 message="illegalOperationFault" />
8608 </wsdl:operation>
8609
8610 <wsdl:operation name="batchComplete">
8611 <wsdl:input message="batchComplete" />
8612 <wsdl:output message="batchCompleteResponse" />
8613 </wsdl:operation>
8614
8615 <wsdl:operation name="remove">
8616 <wsdl:input message="remove" />
8617 <wsdl:output message="removeResponse" />
8618 <wsdl:fault name="illegalArgumentFault"
8619 message="illegalArgumentFault" />
8620 <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8621 <wsdl:fault name="illegalOperationFault"
8622 message="illegalOperationFault" />
8623 </wsdl:operation>
8624
8625 <wsdl:operation name="batchRemove">
8626 <wsdl:input message="batchRemove" />
8627 <wsdl:output message="batchRemoveResponse" />
8628 </wsdl:operation>
8629
8630 <wsdl:operation name="fail">
8631 <wsdl:input message="fail" />
8632 <wsdl:output message="failResponse" />
8633 <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
8634 <wsdl:fault name="illegalArgumentFault"
8635 message="illegalArgumentFault" />
8636 <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />

```

```

8637 -----<wsdl:fault name="illegalOperationFault"
8638 message="illegalOperationFault"/>
8639 -----</wsdl:operation>
8640
8641 -----<wsdl:operation name="batchFail">
8642 -----<wsdl:input message="batchFail"/>
8643 -----<wsdl:output message="batchFailResponse"/>
8644 -----</wsdl:operation>
8645
8646 -----<wsdl:operation name="setPriority">
8647 -----<wsdl:input message="setPriority"/>
8648 -----<wsdl:output message="setPriorityResponse"/>
8649 -----<wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
8650 -----<wsdl:fault name="illegalArgumentFault"
8651 message="illegalArgumentFault"/>
8652 -----<wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
8653 -----<wsdl:fault name="illegalOperationFault"
8654 message="illegalOperationFault"/>
8655 -----</wsdl:operation>
8656
8657 -----<wsdl:operation name="batchSetPriority">
8658 -----<wsdl:input message="batchSetPriority"/>
8659 -----<wsdl:output message="batchSetPriorityResponse"/>
8660 -----</wsdl:operation>
8661
8662 -----updateComment">
8663 -----<wsdl:input message="updateComment"/>
8664 -----<wsdl:output message="updateCommentResponse"/>
8665 -----<wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
8666 -----<wsdl:fault name="illegalArgumentFault"
8667 message="illegalArgumentFault"/>
8668 -----<wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
8669 -----<wsdl:fault name="illegalOperationFault"
8670 message="illegalOperationFault"/>
8671 -----</wsdl:operation>
8672
8673 -----<wsdl:operation name="addAttachment">
8674 -----<wsdl:input message="addAttachment"/>
8675 -----<wsdl:output message="addAttachmentResponse"/>
8676 -----<wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
8677 -----<wsdl:fault name="illegalArgumentFault"
8678 message="illegalArgumentFault"/>
8679 -----<wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
8680 -----<wsdl:fault name="illegalOperationFault"
8681 message="illegalOperationFault"/>
8682 -----</wsdl:operation>
8683
8684 -----<wsdl:operation name="getAttachmentInfos">
8685 -----<wsdl:input message="getAttachmentInfos"/>
8686 -----<wsdl:output message="getAttachmentInfosResponse"/>
8687 -----<wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
8688 -----<wsdl:fault name="illegalArgumentFault"
8689 message="illegalArgumentFault"/>
8690 -----<wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
8691 -----<wsdl:fault name="illegalOperationFault"
8692 message="illegalOperationFault"/>
8693 -----</wsdl:operation>
8694

```



```

8695 <wsdl:operation name="getAttachment">
8696 <wsdl:input message="getAttachment"/>
8697 <wsdl:output message="getAttachmentResponse"/>
8698 <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
8699 <wsdl:fault name="illegalArgumentFault"
8700 message="illegalArgumentFault"/>
8701 <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
8702 <wsdl:fault name="illegalOperationFault"
8703 message="illegalOperationFault"/>
8704 </wsdl:operation>
8705
8706 <wsdl:operation name="deleteAttachment">
8707 <wsdl:input message="deleteAttachment"/>
8708 <wsdl:output message="deleteAttachmentResponse"/>
8709 <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
8710 <wsdl:fault name="illegalArgumentFault"
8711 message="illegalArgumentFault"/>
8712 <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
8713 <wsdl:fault name="illegalOperationFault"
8714 message="illegalOperationFault"/>
8715 </wsdl:operation>
8716
8717 <wsdl:operation name="addComment">
8718 <wsdl:input message="addComment"/>
8719 <wsdl:output message="addCommentResponse"/>
8720 <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
8721 <wsdl:fault name="illegalArgumentFault"
8722 message="illegalArgumentFault"/>
8723 <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
8724 <wsdl:fault name="illegalOperationFault"
8725 message="illegalOperationFault"/>
8726 </wsdl:operation>
8727
8728 <wsdl:operation name="getComments">
8729 <wsdl:input message="getComments"/>
8730 <wsdl:output message="getCommentsResponse"/>
8731 <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
8732 <wsdl:fault name="illegalArgumentFault"
8733 message="illegalArgumentFault"/>
8734 <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
8735 <wsdl:fault name="illegalOperationFault"
8736 message="illegalOperationFault"/>
8737 </wsdl:operation>
8738
8739 <wsdl:operation name="skip">
8740 <wsdl:input message="skip"/>
8741 <wsdl:output message="skipResponse"/>
8742 <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
8743 <wsdl:fault name="illegalArgumentFault"
8744 message="illegalArgumentFault"/>
8745 <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
8746 <wsdl:fault name="illegalOperationFault"
8747 message="illegalOperationFault"/>
8748 </wsdl:operation>
8749
8750 <wsdl:operation name="batchSkip">
8751 <wsdl:input message="batchSkip"/>
8752 <wsdl:output message="batchSkipResponse"/>

```

```

8753 </wsdl:operation>
8754
8755 <wsdl:operation name="forward">
8756 <wsdl:input message="forward"/>
8757 <wsdl:output message="forwardResponse"/>
8758 <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
8759 <wsdl:fault name="illegalArgumentFault"
8760 message="illegalArgumentFault"/>
8761 <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
8762 <wsdl:fault name="illegalOperationFault"
8763 message="illegalOperationFault"/>
8764 </wsdl:operation>
8765
8766 <wsdl:operation name="batchForward">
8767 <wsdl:input message="batchForward"/>
8768 <wsdl:output message="batchForwardResponse"/>
8769 </wsdl:operation>
8770
8771 <wsdl:operation name="delegate">
8772 <wsdl:input message="delegate"/>
8773 <wsdl:output message="delegateResponse"/>
8774 <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
8775 <wsdl:fault name="illegalArgumentFault"
8776 message="illegalArgumentFault"/>
8777 <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
8778 <wsdl:fault name="illegalOperationFault"
8779 message="illegalOperationFault"/>
8780 <wsdl:fault name="recipientNotAllowed" message="recipientNotAllowed"/>
8781 </wsdl:operation>
8782
8783 <wsdl:operation name="batchDelegate">
8784 <wsdl:input message="batchDelegate"/>
8785 <wsdl:output message="batchDelegateResponse"/>
8786 </wsdl:operation>
8787
8788 <wsdl:operation name="getRendering">
8789 <wsdl:input message="getRendering"/>
8790 <wsdl:output message="getRenderingResponse"/>
8791 <wsdl:fault name="illegalArgumentFault"
8792 message="illegalArgumentFault"/>
8793 </wsdl:operation>
8794
8795 <wsdl:operation name="getRenderingTypes">
8796 <wsdl:input message="getRenderingTypes"/>
8797 <wsdl:output message="getRenderingTypesResponse"/>
8798 <wsdl:fault name="illegalArgumentFault"
8799 message="illegalArgumentFault"/>
8800 </wsdl:operation>
8801
8802 <wsdl:operation name="getTaskDetails">
8803 <wsdl:input message="getTaskDetails"/>
8804 <wsdl:output message="getTaskDetailsResponse"/>
8805 <wsdl:fault name="illegalArgumentFault"
8806 message="illegalArgumentFault"/>
8807 </wsdl:operation>
8808
8809 <wsdl:operation name="getTaskDescription">
8810 <wsdl:input message="getTaskDescription"/>

```

```

8811 -----<wsdl:output message="getTaskDescriptionResponse"/>
8812 -----<wsdl:fault name="illegalArgumentFault"
8813 message="illegalArgumentFault"/>
8814 -----</wsdl:operation>
8815
8816 -----<wsdl:operation name="setOutput">
8817 -----<wsdl:input message="setOutput"/>
8818 -----<wsdl:output message="setOutputResponse"/>
8819 -----<wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
8820 -----<wsdl:fault name="illegalArgumentFault"
8821 message="illegalArgumentFault"/>
8822 -----<wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
8823 -----<wsdl:fault name="illegalOperationFault"
8824 message="illegalOperationFault"/>
8825 -----</wsdl:operation>
8826
8827 -----<wsdl:operation name="deleteOutput">
8828 -----<wsdl:input message="deleteOutput"/>
8829 -----<wsdl:output message="deleteOutputResponse"/>
8830 -----<wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
8831 -----<wsdl:fault name="illegalArgumentFault"
8832 message="illegalArgumentFault"/>
8833 -----<wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
8834 -----<wsdl:fault name="illegalOperationFault"
8835 message="illegalOperationFault"/>
8836 -----</wsdl:operation>
8837
8838 -----<wsdl:operation name="setFault">
8839 -----<wsdl:input message="setFault"/>
8840 -----<wsdl:output message="setFaultResponse"/>
8841 -----<wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
8842 -----<wsdl:fault name="illegalArgumentFault"
8843 message="illegalArgumentFault"/>
8844 -----<wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
8845 -----<wsdl:fault name="illegalOperationFault"
8846 message="illegalOperationFault"/>
8847 -----</wsdl:operation>
8848
8849 -----<wsdl:operation name="deleteFault">
8850 -----<wsdl:input message="deleteFault"/>
8851 -----<wsdl:output message="deleteFaultResponse"/>
8852 -----<wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
8853 -----<wsdl:fault name="illegalArgumentFault"
8854 message="illegalArgumentFault"/>
8855 -----<wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
8856 -----<wsdl:fault name="illegalOperationFault"
8857 message="illegalOperationFault"/>
8858 -----</wsdl:operation>
8859
8860 -----<wsdl:operation name="getInput">
8861 -----<wsdl:input message="getInput"/>
8862 -----<wsdl:output message="getInputResponse"/>
8863 -----<wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
8864 -----<wsdl:fault name="illegalArgumentFault"
8865 message="illegalArgumentFault"/>
8866 -----<wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
8867 -----<wsdl:fault name="illegalOperationFault"
8868 message="illegalOperationFault"/>

```

```

8869 </wsdl:operation>
8870
8871 <wsdl:operation name="getOutput">
8872 <wsdl:input message="getOutput"/>
8873 <wsdl:output message="getOutputResponse"/>
8874 <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
8875 <wsdl:fault name="illegalArgumentFault"
8876 message="illegalArgumentFault"/>
8877 <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
8878 <wsdl:fault name="illegalOperationFault"
8879 message="illegalOperationFault"/>
8880 </wsdl:operation>
8881
8882 <wsdl:operation name="getFault">
8883 <wsdl:input message="getFault"/>
8884 <wsdl:output message="getFaultResponse"/>
8885 <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
8886 <wsdl:fault name="illegalArgumentFault"
8887 message="illegalArgumentFault"/>
8888 <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
8889 <wsdl:fault name="illegalOperationFault"
8890 message="illegalOperationFault"/>
8891 </wsdl:operation>
8892
8893 <wsdl:operation name="getMyTaskAbstracts">
8894 <wsdl:input message="getMyTaskAbstracts"/>
8895 <wsdl:output message="getMyTaskAbstractsResponse"/>
8896 <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
8897 <wsdl:fault name="illegalArgumentFault"
8898 message="illegalArgumentFault"/>
8899 <wsdl:fault name="illegalOperationFault"
8900 message="illegalOperationFault"/>
8901 </wsdl:operation>
8902
8903 <wsdl:operation name="getMyTaskDetails">
8904 <wsdl:input message="getMyTaskDetails"/>
8905 <wsdl:output message="getMyTaskDetailsResponse"/>
8906 <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
8907 <wsdl:fault name="illegalArgumentFault"
8908 message="illegalArgumentFault"/>
8909 <wsdl:fault name="illegalOperationFault"
8910 message="illegalOperationFault"/>
8911 </wsdl:operation>
8912
8913 <wsdl:operation name="query">
8914 <wsdl:input message="query"/>
8915 <wsdl:output message="queryResponse"/>
8916 <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
8917 <wsdl:fault name="illegalArgumentFault"
8918 message="illegalArgumentFault"/>
8919 </wsdl:operation>
8920
8921 <wsdl:operation name="activate">
8922 <wsdl:input message="activate"/>
8923 <wsdl:output message="activateResponse"/>
8924 <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
8925 <wsdl:fault name="illegalArgumentFault"
8926 message="illegalArgumentFault"/>

```

```

8927     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8928     <wsdl:fault name="illegalOperationFault"
8929 message="illegalOperationFault" />
8930   </wsdl:operation>
8931
8932   <wsdl:operation name="batchActivate">
8933     <wsdl:input message="batchActivate" />
8934     <wsdl:output message="batchActivateResponse" />
8935   </wsdl:operation>
8936
8937   <wsdl:operation name="nominate">
8938     <wsdl:input message="nominate" />
8939     <wsdl:output message="nominateResponse" />
8940     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
8941     <wsdl:fault name="illegalArgumentFault"
8942 message="illegalArgumentFault" />
8943     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8944     <wsdl:fault name="illegalOperationFault"
8945 message="illegalOperationFault" />
8946   </wsdl:operation>
8947
8948   <wsdl:operation name="batchNominate">
8949     <wsdl:input message="batchNominate" />
8950     <wsdl:output message="batchNominateResponse" />
8951   </wsdl:operation>
8952
8953   <wsdl:operation name="setGenericHumanRole">
8954     <wsdl:input message="setGenericHumanRole" />
8955     <wsdl:output message="setGenericHumanRoleResponse" />
8956     <wsdl:fault name="illegalStateFault" message="illegalStateFault" />
8957     <wsdl:fault name="illegalArgumentFault"
8958 message="illegalArgumentFault" />
8959     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault" />
8960     <wsdl:fault name="illegalOperationFault"
8961 message="illegalOperationFault" />
8962   </wsdl:operation>
8963
8964   <wsdl:operation name="batchSetGenericHumanRole">
8965     <wsdl:input message="batchSetGenericHumanRole" />
8966     <wsdl:output message="batchSetGenericHumanRoleResponse" />
8967   </wsdl:operation>
8968
8969   <wsdl:operation name="getOutcome">
8970   <wsdl:input message="getOutcome" />
8971   <wsdl:output message="getOutcomeResponse" />
8972   <wsdl:fault name="illegalArgumentFault"
8973   message="illegalArgumentFault" />
8974   <wsdl:fault name="illegalOperationFault"
8975   message="illegalOperationFault" />
8976   </wsdl:operation>
8977
8978   <wsdl:operation name="getTaskOperations">
8979   <wsdl:input message="getTaskOperations" />
8980   <wsdl:output message="getTaskOperationsResponse" />
8981   <wsdl:fault name="illegalArgumentFault"
8982   message="illegalArgumentFault" />
8983   <wsdl:fault name="illegalOperationFault"
8984   message="illegalOperationFault" />

```

```

8985 -----</wsdl:operation>
8986
8987 -----<wsdl:operation name="getTaskInstanceData">
8988 -----<wsdl:input message="getTaskInstanceData"/>
8989 -----<wsdl:output message="getTaskInstanceDataResponse"/>
8990 -----<wsdl:fault name="illegalArgumentFault"
8991 message="illegalArgumentFault"/>
8992 -----<wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
8993 -----<wsdl:fault name="illegalOperationFault"
8994 message="illegalOperationFault"/>
8995 -----</wsdl:operation>
8996
8997 -----<wsdl:operation name="getTaskHistory">
8998 -----<wsdl:input message="getTaskHistory"/>
8999 -----<wsdl:output message="getTaskHistoryResponse"/>
9000 -----<wsdl:fault name="illegalArgumentFault"
9001 message="illegalArgumentFault"/>
9002 -----<wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
9003 -----<wsdl:fault name="illegalOperationFault"
9004 message="illegalOperationFault"/>
9005 -----</wsdl:operation>
9006
9007 -----<wsdl:operation name="setTaskStartDeadlineExpression">
9008 -----<wsdl:input message="setTaskStartDeadlineExpression"/>
9009 -----<wsdl:output message="setTaskStartDeadlineExpressionResponse"/>
9010 -----<wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
9011 -----<wsdl:fault name="illegalArgumentFault"
9012 message="illegalArgumentFault"/>
9013 -----<wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
9014 -----<wsdl:fault name="illegalOperationFault"
9015 message="illegalOperationFault"/>
9016 -----</wsdl:operation>
9017
9018 -----<wsdl:operation name="setTaskStartDurationExpression">
9019 -----<wsdl:input message="setTaskStartDurationExpression"/>
9020 -----<wsdl:output message="setTaskStartDurationExpressionResponse"/>
9021 -----<wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
9022 -----<wsdl:fault name="illegalArgumentFault"
9023 message="illegalArgumentFault"/>
9024 -----<wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
9025 -----<wsdl:fault name="illegalOperationFault"
9026 message="illegalOperationFault"/>
9027 -----</wsdl:operation>
9028
9029 -----<wsdl:operation name="setTaskCompletionDeadlineExpression">
9030 -----<wsdl:input message="setTaskCompletionDeadlineExpression"/>
9031 -----<wsdl:output message="setTaskCompletionDeadlineExpressionResponse"/>
9032 -----<wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
9033 -----<wsdl:fault name="illegalArgumentFault"
9034 message="illegalArgumentFault"/>
9035 -----<wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
9036 -----<wsdl:fault name="illegalOperationFault"
9037 message="illegalOperationFault"/>
9038 -----</wsdl:operation>
9039
9040 -----<wsdl:operation name="setTaskCompletionDurationExpression">
9041 -----<wsdl:input message="setTaskCompletionDurationExpression"/>
9042 -----<wsdl:output message="setTaskCompletionDurationExpressionResponse"/>

```

```
9043 <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
9044 <wsdl:fault name="illegalArgumentFault"
9045 message="illegalArgumentFault"/>
9046 <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
9047 <wsdl:fault name="illegalOperationFault"
9048 message="illegalOperationFault"/>
9049 </wsdl:operation>
9050
9051 </wsdl:portType>
9052 </wsdl:definitions>
```


9053

E. WS-HumanTask Parent API Port Type

```
9054 <?xml version="1.0" encoding="UTF-8"?>
9055 <!--
9056 Copyright (c) OASIS Open 2009. All Rights Reserved.
9057 -->
9058 <wsdl:definitions
9059     targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
9060 humantask/leantask/api/200803"
9061     xmlns="http://docs.oasis-open.org/ns/bpel4people/ws-
9062 humantask/leantask/api/200803"
9063     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
9064     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
9065     xmlns:htd="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803"
9066     xmlns:htt="http://docs.oasis-open.org/ns/bpel4people/ws-
9067 humantask/types/200803">
9068
9069     <wsdl:documentation>
9070         Web Service Definition for WS-HumanTask 1.1 - Operations for Task Parent
9071 Applications
9072     </wsdl:documentation>
9073
9074     <wsdl:types>
9075         <xsd:schema
9076             targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
9077 humantask/leantask/api/200803"
9078             elementFormDefault="qualified"
9079             blockDefault="#all">
9080
9081             <xsd:import
9082                 namespace="http://docs.oasis-open.org/ns/bpel4people/ws-
9083 humantask/200803"
9084                 schemaLocation="ws-humantask.xsd" />
9085             <xsd:import
9086                 namespace="http://docs.oasis-open.org/ns/bpel4people/ws-
9087 humantask/types/200803"
9088                 schemaLocation="ws-humantask-types.xsd" />
9089
9090             <!-- Input and output elements -->
9091             <xsd:element name="registerLeanTaskDefinition">
9092                 <xsd:complexType>
9093                     <xsd:sequence>
9094                         <xsd:element name="taskDefinition" type="htd:tLeanTask" />
9095                     </xsd:sequence>
9096                 </xsd:complexType>
9097             </xsd:element>
9098             <xsd:element name="registerLeanTaskDefinitionResponse">
9099                 <xsd:complexType>
9100                     <xsd:sequence>
9101                         <xsd:element name="taskName" type="xsd:NCName" />
9102                     </xsd:sequence>
9103                 </xsd:complexType>
9104             </xsd:element>
9105
9106             <xsd:element name="unregisterLeanTaskDefinition">
```



```

9107     <xsd:complexType>
9108         <xsd:sequence>
9109             <xsd:element name="taskName" type="xsd:NCName" />
9110         </xsd:sequence>
9111     </xsd:complexType>
9112 </xsd:element>
9113 <xsd:element name="unregisterLeanTaskDefinitionResponse">
9114     <xsd:complexType>
9115         <xsd:sequence>
9116             <xsd:element name="taskName" type="xsd:NCName" />
9117         </xsd:sequence>
9118     </xsd:complexType>
9119 </xsd:element>
9120
9121 <xsd:element name="listLeanTaskDefinitions">
9122     <xsd:complexType>
9123         <xsd:sequence>
9124             <xsd:annotation>
9125                 <xsd:documentation>Empty message</xsd:documentation>
9126             </xsd:annotation>
9127         </xsd:sequence>
9128     </xsd:complexType>
9129 </xsd:element>
9130 <xsd:element name="listLeanTaskDefinitionsResponse">
9131     <xsd:complexType>
9132         <xsd:sequence>
9133             <xsd:element name="leanTaskDefinitions">
9134                 <xsd:complexType>
9135                     <xsd:sequence>
9136                         <xsd:element name="leanTaskDefinition" type="htd:tLeanTask"
9137 minOccurs="0" maxOccurs="unbounded" />
9138                     </xsd:sequence>
9139                 </xsd:complexType>
9140             </xsd:element>
9141         </xsd:sequence>
9142     </xsd:complexType>
9143 </xsd:element>
9144
9145 <xsd:element name="createLeanTask">
9146     <xsd:complexType>
9147         <xsd:sequence>
9148             <xsd:element name="inputMessage">
9149                 <xsd:complexType>
9150                     <xsd:sequence>
9151                         <xsd:any processContents="lax" namespace="##any" />
9152                     </xsd:sequence>
9153                 </xsd:complexType>
9154             </xsd:element>
9155             <xsd:element name="taskDefinition" type="htd:tLeanTask"
9156 minOccurs="0"/>
9157                 <xsd:element name="taskName" type="xsd:NCName" minOccurs="0" />
9158             </xsd:sequence>
9159         </xsd:complexType>
9160     </xsd:element>
9161 <xsd:element name="createLeanTaskResponse">
9162     <xsd:complexType>
9163         <xsd:sequence>
9164             <xsd:element name="outputMessage">

```

```

9165         <xsd:complexType>
9166             <xsd:sequence>
9167                 <xsd:any processContents="lax" namespace="##any" />
9168             </xsd:sequence>
9169         </xsd:complexType>
9170     </xsd:element>
9171 </xsd:sequence>
9172 </xsd:complexType>
9173 </xsd:element>
9174
9175 <xsd:element name="createLeanTaskAsync">
9176     <xsd:complexType>
9177         <xsd:sequence>
9178             <xsd:element name="inputMessage">
9179                 <xsd:complexType>
9180                     <xsd:sequence>
9181                         <xsd:any processContents="lax" namespace="##any" />
9182                     </xsd:sequence>
9183                 </xsd:complexType>
9184             </xsd:element>
9185             <xsd:element name="taskDefinition" type="htd:tLeanTask"
9186 minOccurs="0"/>
9187             <xsd:element name="taskName" type="xsd:NCName" minOccurs="0" />
9188         </xsd:sequence>
9189     </xsd:complexType>
9190 </xsd:element>
9191 <xsd:element name="createLeanTaskAsyncResponse">
9192     <xsd:complexType>
9193         <xsd:sequence>
9194             <xsd:annotation>
9195                 <xsd:documentation>Empty message</xsd:documentation>
9196             </xsd:annotation>
9197         </xsd:sequence>
9198     </xsd:complexType>
9199 </xsd:element>
9200
9201 <xsd:element name="createLeanTaskAsyncCallback">
9202     <xsd:complexType>
9203         <xsd:sequence>
9204             <xsd:element name="outputMessage">
9205                 <xsd:complexType>
9206                     <xsd:sequence>
9207                         <xsd:any processContents="lax" namespace="##any" />
9208                     </xsd:sequence>
9209                 </xsd:complexType>
9210             </xsd:element>
9211         </xsd:sequence>
9212     </xsd:complexType>
9213 </xsd:element>
9214
9215 <!-- Fault elements -->
9216 <xsd:element name="illegalState">
9217     <xsd:complexType>
9218         <xsd:sequence>
9219             <xsd:element name="status" type="htt:tStatus"/>
9220             <xsd:element name="message" type="xsd:string"/>
9221         </xsd:sequence>
9222     </xsd:complexType>

```

```

9223     </xsd:element>
9224
9225     <xsd:element name="illegalArgument" type="xsd:string"/>
9226
9227     <xsd:element name="illegalAccess" type="xsd:string"/>
9228
9229     </xsd:schema>
9230 </wsdl:types>
9231
9232 <!-- Declaration of messages -->
9233 <wsdl:message name="registerLeanTaskDefinition">
9234   <wsdl:part name="registerLeanTaskDefinition"
9235 element="registerLeanTaskDefinition"/>
9236 </wsdl:message>
9237 <wsdl:message name="registerLeanTaskDefinitionResponse">
9238   <wsdl:part name="registerLeanTaskDefinitionResponse"
9239 element="registerLeanTaskDefinitionResponse"/>
9240 </wsdl:message>
9241
9242 <wsdl:message name="unregisterLeanTaskDefinition">
9243   <wsdl:part name="unregisterLeanTaskDefinition"
9244 element="unregisterLeanTaskDefinition"/>
9245 </wsdl:message>
9246 <wsdl:message name="unregisterLeanTaskDefinitionResponse">
9247   <wsdl:part name="unregisterLeanTaskDefinitionResponse"
9248 element="unregisterLeanTaskDefinitionResponse"/>
9249 </wsdl:message>
9250
9251 <wsdl:message name="listLeanTaskDefinitions">
9252   <wsdl:part name="listLeanTaskDefinitions"
9253 element="listLeanTaskDefinitions"/>
9254 </wsdl:message>
9255 <wsdl:message name="listLeanTaskDefinitionsResponse">
9256   <wsdl:part name="listLeanTaskDefinitionsResponse"
9257 element="listLeanTaskDefinitionsResponse"/>
9258 </wsdl:message>
9259
9260 <wsdl:message name="createLeanTask">
9261   <wsdl:part name="createLeanTask" element="createLeanTask"/>
9262 </wsdl:message>
9263 <wsdl:message name="createLeanTaskResponse">
9264   <wsdl:part name="createLeanTaskResponse"
9265 element="createLeanTaskResponse"/>
9266 </wsdl:message>
9267
9268 <wsdl:message name="createLeanTaskAsync">
9269   <wsdl:part name="createLeanTaskAsync" element="createLeanTaskAsync"/>
9270 </wsdl:message>
9271 <wsdl:message name="createLeanTaskAsyncResponse">
9272   <wsdl:part name="createLeanTaskAsyncResponse"
9273 element="createLeanTaskAsyncResponse"/>
9274 </wsdl:message>
9275
9276 <wsdl:message name="createLeanTaskAsyncCallback">
9277   <wsdl:part name="createLeanTaskAsyncCallback"
9278 element="createLeanTaskAsyncCallback"/>
9279 </wsdl:message>
9280

```

```

9281 <!-- Declaration of fault messages -->
9282 <wsdl:message name="illegalStateFault">
9283   <wsdl:part name="illegalState" element="illegalState"/>
9284 </wsdl:message>
9285 <wsdl:message name="illegalArgumentFault">
9286   <wsdl:part name="illegalArgument" element="illegalArgument"/>
9287 </wsdl:message>
9288 <wsdl:message name="illegalAccessFault">
9289   <wsdl:part name="illegalAccess" element="illegalAccess"/>
9290 </wsdl:message>
9291
9292 <!-- Port type definitions -->
9293 <wsdl:portType name="leanTaskOperations">
9294
9295   <wsdl:operation name="registerLeanTaskDefinition">
9296     <wsdl:input message="registerLeanTaskDefinition"/>
9297     <wsdl:output message="registerLeanTaskDefinitionResponse"/>
9298     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
9299     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
9300   </wsdl:operation>
9301
9302   <wsdl:operation name="unregisterLeanTaskDefinition">
9303     <wsdl:input message="unregisterLeanTaskDefinition"/>
9304     <wsdl:output message="unregisterLeanTaskDefinitionResponse"/>
9305     <wsdl:fault name="illegalArgumentFault"
9306 message="illegalArgumentFault"/>
9307     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
9308   </wsdl:operation>
9309
9310   <wsdl:operation name="listLeanTaskDefinitions">
9311     <wsdl:input message="listLeanTaskDefinitions"/>
9312     <wsdl:output message="listLeanTaskDefinitionsResponse"/>
9313     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
9314   </wsdl:operation>
9315
9316   <wsdl:operation name="createLeanTask">
9317     <wsdl:input message="createLeanTask"/>
9318     <wsdl:output message="createLeanTaskResponse"/>
9319     <wsdl:fault name="illegalArgumentFault"
9320 message="illegalArgumentFault"/>
9321     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
9322   </wsdl:operation>
9323
9324   <wsdl:operation name="createLeanTaskAsync">
9325     <wsdl:input message="createLeanTaskAsync"/>
9326     <wsdl:output message="createLeanTaskAsyncResponse"/>
9327     <wsdl:fault name="illegalArgumentFault"
9328 message="illegalArgumentFault"/>
9329     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
9330   </wsdl:operation>
9331
9332 </wsdl:portType>
9333
9334 <wsdl:portType name="leanTaskCallbackOperations">
9335
9336   <wsdl:operation name="createLeanTaskAsyncCallback">
9337     <wsdl:input message="createLeanTaskAsyncCallback"/>
9338   </wsdl:operation>

```

```
9339
9340     </wsdl:portType>
9341
9342 </wsdl:definitions>
```

9343

F. WS-HumanTask Protocol Handler Port Types

```

9344 <?xml version="1.0" encoding="UTF-8"?>
9345 <!--
9346 Copyright (c) OASIS Open 2009. All Rights Reserved.
9347 -->
9348 <wsdl:definitions
9349     targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
9350 humantask/protocol/200803"
9351     xmlns="http://docs.oasis-open.org/ns/bpel4people/ws-
9352 humantask/protocol/200803"
9353     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
9354     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
9355     xmlns:htp="http://docs.oasis-open.org/ns/bpel4people/ws-
9356 humantask/protocol/200803">
9357
9358     <wsdl:documentation>
9359         Web Service Definition for WS-HumanTask 1.1 - Operations WS-HumanTask
9360 Protocol Participants
9361     </wsdl:documentation>
9362
9363     <wsdl:types>
9364     <xsd:schema
9365         targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
9366 humantask/protocol/200803"
9367         elementFormDefault="qualified"
9368         blockDefault="#all">
9369
9370         <xsd:complexType name="tProtocolMsgType">
9371         <xsd:sequence>
9372             <xsd:any namespace="##other" processContents="lax" minOccurs="0"
9373                 maxOccurs="unbounded" />
9374         </xsd:sequence>
9375         <xsd:anyAttribute namespace="##any" processContents="lax" />
9376         </xsd:complexType>
9377
9378         <xsd:element name="skipped" type="htp:tProtocolMsgType" />
9379         <xsd:element name="fault" type="htp:tProtocolMsgType" />
9380         <xsd:element name="exit" type="htp:tProtocolMsgType" />
9381
9382         <xsd:element name="responseAction" type="xsd:anyURI" />
9383         <xsd:element name="responseOperation" type="xsd:NCName" />
9384
9385     </xsd:schema>
9386 </wsdl:types>
9387
9388     <wsdl:message name="skipped">
9389         <wsdl:part name="parameters" element="skipped" />
9390     </wsdl:message>
9391     <wsdl:message name="fault">
9392         <wsdl:part name="parameters" element="fault" />
9393     </wsdl:message>
9394     <wsdl:message name="exit">
9395         <wsdl:part name="parameters" element="exit" />
9396     </wsdl:message>

```

```
9397
9398 <wsdl:portType name="clientParticipantPortType">
9399   <wsdl:operation name="skippedOperation">
9400     <wsdl:input message="skipped" />
9401   </wsdl:operation>
9402   <wsdl:operation name="faultOperation">
9403     <wsdl:input message="fault" />
9404   </wsdl:operation>
9405 </wsdl:portType>
9406
9407 <wsdl:portType name="humanTaskParticipantPortType">
9408   <wsdl:operation name="exitOperation">
9409     <wsdl:input message="exit" />
9410   </wsdl:operation>
9411 </wsdl:portType>
9412
9413 </wsdl:definitions>
```

9414

G. WS-HumanTask Context Schema

```

9415 <?xml version="1.0" encoding="UTF-8"?>
9416 <!--
9417   Copyright (c) OASIS Open 2009. All Rights Reserved.
9418 -->
9419 <xsd:schema
9420   targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
9421   humantask/context/200803"
9422   xmlns="http://docs.oasis-open.org/ns/bpel4people/ws-
9423   humantask/context/200803"
9424   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
9425   xmlns:htt="http://docs.oasis-open.org/ns/bpel4people/ws-
9426   humantask/types/200803"
9427   elementFormDefault="qualified"
9428   blockDefault="#all">
9429
9430   <xsd:annotation>
9431     <xsd:documentation>
9432       XML Schema for WS-HumanTask 1.1 - Human Task Context for Task
9433       Interactions
9434     </xsd:documentation>
9435   </xsd:annotation>
9436
9437   <!-- other namespaces -->
9438   <xsd:import
9439     namespace="http://www.w3.org/XML/1998/namespace"
9440     schemaLocation="http://www.w3.org/2001/xml.xsd" />
9441   <xsd:import
9442     namespace="http://docs.oasis-open.org/ns/bpel4people/ws-
9443     humantask/types/200803"
9444     schemaLocation="ws-humantask-types.xsd" />
9445
9446   <!-- human task context -->
9447   <xsd:element name="humanTaskRequestContext"
9448   type="tHumanTaskRequestContext" />
9449   <xsd:complexType name="tHumanTaskRequestContext">
9450     <xsd:complexContent>
9451       <xsd:extension base="tHumanTaskContextBase">
9452         <xsd:sequence>
9453           <xsd:element name="peopleAssignments" type="tPeopleAssignments"
9454   minOccurs="0" />
9455           <xsd:element name="isSkipable" type="xsd:boolean" minOccurs="0" />
9456           <xsd:element name="expirationTime" type="xsd:dateTime"
9457   minOccurs="0" />
9458           <xsd:element name="activationDeferralTime" type="xsd:dateTime"
9459   minOccurs="0" />
9460           <xsd:any namespace="##other" processContents="lax" minOccurs="0"
9461   maxOccurs="unbounded" />
9462         </xsd:sequence>
9463       </xsd:extension>
9464     </xsd:complexContent>
9465   </xsd:complexType>
9466   <xsd:element name="humanTaskResponseContext"
9467   type="tHumanTaskResponseContext" />

```



```

9468 <xsd:complexType name="tHumanTaskResponseContext">
9469 <xsd:complexContent>
9470 <xsd:extension base="tHumanTaskContextBase">
9471 <xsd:sequence>
9472 <xsd:element name="actualOwner" type="htt:tUser"/>
9473 <xsd:element name="actualPeopleAssignments"
9474 type="tPeopleAssignments"/>
9475 <xsd:element name="outcome" type="xsd:string" minOccurs="0"/>
9476 <xsd:any namespace="##other" processContents="lax" minOccurs="0"
9477 maxOccurs="unbounded"/>
9478 </xsd:sequence>
9479 </xsd:extension>
9480 </xsd:complexContent>
9481 </xsd:complexType>
9482 <xsd:complexType name="tHumanTaskContextBase" abstract="true">
9483 <xsd:sequence>
9484 <xsd:element name="priority" type="htt:tPriority" minOccurs="0"/>
9485 <xsd:element name="attachments" type="tAttachments" minOccurs="0"/>
9486 </xsd:sequence>
9487 </xsd:complexType>
9488
9489 <!-- people assignments -->
9490 <xsd:complexType name="tPeopleAssignments">
9491 <xsd:sequence>
9492 <xsd:element ref="genericHumanRole" minOccurs="0"
9493 maxOccurs="unbounded"/>
9494 </xsd:sequence>
9495 </xsd:complexType>
9496 <xsd:element name="genericHumanRole" type="tGenericHumanRole"
9497 abstract="true" block="restriction extension"/>
9498 <xsd:element name="potentialOwners" type="tGenericHumanRole"
9499 substitutionGroup="genericHumanRole"/>
9500 <xsd:element name="excludedOwners" type="tGenericHumanRole"
9501 substitutionGroup="genericHumanRole"/>
9502 <xsd:element name="taskInitiator" type="tGenericHumanRole"
9503 substitutionGroup="genericHumanRole"/>
9504 <xsd:element name="taskStakeholders" type="tGenericHumanRole"
9505 substitutionGroup="genericHumanRole"/>
9506 <xsd:element name="businessAdministrators" type="tGenericHumanRole"
9507 substitutionGroup="genericHumanRole"/>
9508 <xsd:element name="recipients" type="tGenericHumanRole"
9509 substitutionGroup="genericHumanRole"/>
9510 <xsd:complexType name="tGenericHumanRole">
9511 <xsd:sequence>
9512 <xsd:element ref="htt:organizationalEntity"/>
9513 </xsd:sequence>
9514 </xsd:complexType>
9515
9516 <!-- attachments -->
9517 <xsd:complexType name="tAttachments">
9518 <xsd:sequence>
9519 <xsd:element name="returnAttachments" type="tReturnAttachments"
9520 minOccurs="0"/>
9521 <xsd:element ref="htt:attachment" minOccurs="0" maxOccurs="unbounded"/>
9522 </xsd:sequence>
9523 </xsd:complexType>
9524 <xsd:simpleType name="tReturnAttachments">
9525 <xsd:restriction base="xsd:string">

```

```
9526     <xsd:enumeration value="all" />
9527     <xsd:enumeration value="newOnly" />
9528     <xsd:enumeration value="none" />
9529   </xsd:restriction>
9530 </xsd:simpleType>
9531
9532 </xsd:schema>
```

9533

H. WS-HumanTask Policy Assertion Schema

```
9534 <?xml version="1.0" encoding="UTF-8"?>
9535 <!--
9536 Copyright (c) OASIS Open 2009. All Rights Reserved.
9537 -->
9538 <xsd:schema
9539   targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
9540 humantask/policy/200803"
9541   xmlns="http://docs.oasis-open.org/ns/bpel4people/ws-
9542 humantask/policy/200803"
9543   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
9544   xmlns:wsp="http://www.w3.org/ns/ws-policy"
9545   elementFormDefault="qualified"
9546   blockDefault="#all">
9547
9548   <xsd:annotation>
9549     <xsd:documentation>
9550       XML Schema for WS-HumanTask 1.1 - WS-HumanTask Policy Assertion
9551     </xsd:documentation>
9552   </xsd:annotation>
9553
9554   <!-- other namespaces -->
9555   <xsd:import
9556     namespace="http://www.w3.org/ns/ws-policy"
9557     schemaLocation="http://www.w3.org/2007/02/ws-policy.xsd" />
9558
9559   <!-- ws-humantask policy assertion -->
9560   <xsd:element name="HumanTaskAssertion" type="tHumanTaskAssertion"/>
9561   <xsd:complexType name="tHumanTaskAssertion" >
9562     <xsd:attribute ref="wsp:Optional" />
9563     <xsd:anyAttribute namespace="##any" processContents="lax" />
9564   </xsd:complexType>
9565
9566 </xsd:schema>
```

9567

I. Sample

9568

This appendix contains the full sample used in this specification.

9569

9570

WSDL Definition

9571

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

9572

```
<!--
```

9573

```
Copyright (c) OASIS Open 2009. All Rights Reserved.
```

9574

```
-->
```

9575

```
<wsdl:definitions name="ClaimApproval"
```

9576

```
targetNamespace="http://www.example.com/claims"
```

9577

```
xmlns:tns="http://www.example.com/claims"
```

9578

```
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
```

9579

```
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
```

9580

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

9581

```
<wsdl:documentation>
```

9583

```
Example for WS-HumanTask 1.1 - WS-HumanTask Task Interface Definition
```

9584

```
</wsdl:documentation>
```

9585

```
<wsdl:types>
```

9586

```
<xsd:schema
```

9588

```
targetNamespace="http://www.example.com/claims"
```

9589

```
xmlns:tns="http://www.example.com/claims"
```

9590

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

9591

```
elementFormDefault="qualified">
```

9592

```
<xsd:element name="ClaimApprovalData">
```

9593

```
<xsd:complexType>
```

9594

```
<xsd:sequence>
```

9595

```
<xsd:element name="cust">
```

9596

```
<xsd:complexType>
```

9597

```
<xsd:sequence>
```

9598

```
<xsd:element name="id" type="xsd:string">
```

9599

```
</xsd:element>
```

9600

```
<xsd:element name="firstname" type="xsd:string">
```

9601

```
</xsd:element>
```

9602

```
<xsd:element name="lastname" type="xsd:string">
```

9603

```
</xsd:element>
```

9604

```
</xsd:sequence>
```

9605

```
</xsd:complexType>
```

9606

```
</xsd:element>
```

9607

```
<xsd:element name="amount" type="xsd:double" />
```

9608

```
<xsd:element name="region" type="xsd:string" />
```

9609

```
<xsd:element name="prio" type="xsd:int" />
```

9610

```
<xsd:element name="activateAt" type="xsd:dateTime" />
```

9611

```
</xsd:sequence>
```

9612

```
</xsd:complexType>
```

9613

```
</xsd:element>
```

9614

```
</xsd:schema>
```

9615

```
</wsdl:types>
```

9616

```
<wsdl:message name="ClaimApprovalRequest">
```

9618

```
<wsdl:part name="ClaimApprovalRequest"
```

9619

```
element="tns:ClaimApprovalData" />
```

```

9620 </wsdl:message>
9621 <wsdl:message name="ClaimApprovalResponse">
9622   <wsdl:part name="ClaimApprovalResponse" type="xsd:boolean" />
9623 </wsdl:message>
9624 <wsdl:message name="notifyRequest">
9625   <wsdl:part name="firstname" type="xsd:string" />
9626   <wsdl:part name="lastname" type="xsd:string" />
9627 </wsdl:message>
9628
9629 <wsdl:portType name="ClaimsHandlingPT">
9630   <wsdl:operation name="approve">
9631     <wsdl:input message="tns:ClaimApprovalRequest" />
9632   </wsdl:operation>
9633   <wsdl:operation name="escalate">
9634     <wsdl:input message="tns:ClaimApprovalRequest" />
9635   </wsdl:operation>
9636 </wsdl:portType>
9637
9638 <wsdl:portType name="ClaimsHandlingCallbackPT">
9639   <wsdl:operation name="approvalResponse">
9640     <wsdl:input message="tns:ClaimApprovalResponse" />
9641   </wsdl:operation>
9642 </wsdl:portType>
9643
9644 <wsdl:portType name="ClaimApprovalReminderPT">
9645   <wsdl:operation name="notify">
9646     <wsdl:input message="tns:notifyRequest" />
9647   </wsdl:operation>
9648 </wsdl:portType>
9649
9650 </wsdl:definitions>

```

9651
9652 **Human Interaction Definition**

```

9653 <?xml version="1.0" encoding="UTF-8"?>
9654 <!--
9655   Copyright (c) OASIS Open 2009. All Rights Reserved.
9656 -->
9657 <htd:humanInteractions
9658   xmlns:htd="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803"
9659   xmlns:htt="http://docs.oasis-open.org/ns/bpel4people/ws-
9660 humantask/types/200803"
9661   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
9662   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
9663   xmlns:cl="http://www.example.com/claims/"
9664   xmlns:tns="http://www.example.com"
9665   targetNamespace="http://www.example.com"
9666   xsi:schemaLocation="http://docs.oasis-open.org/ns/bpel4people/ws-
9667 humantask/200803 ../../xml/ws-humantask.xsd">
9668
9669   <htd:documentation>
9670     Example for WS-HumanTask 1.1 - WS-HumanTask Task Definition
9671   </htd:documentation>
9672
9673   <htd:import importType="http://schemas.xmlsoap.org/wsdl/"
9674     location="ws-humantask-example-claim-approval.wsdl"
9675     namespace="http://www.example.com/claims/" />
9676

```

```

9677 <htd:logicalPeopleGroups>
9678
9679   <htd:logicalPeopleGroup name="regionalClerks">
9680     <htd:documentation xml:lang="en-US">
9681       The group of clerks responsible for the region specified.
9682     </htd:documentation>
9683     <htd:parameter name="region" type="xsd:string" />
9684   </htd:logicalPeopleGroup>
9685
9686   <htd:logicalPeopleGroup name="regionalManager">
9687     <htd:documentation xml:lang="en-US">
9688       The manager responsible for the region specified.
9689     </htd:documentation>
9690     <htd:parameter name="region" type="xsd:string" />
9691   </htd:logicalPeopleGroup>
9692
9693   <htd:logicalPeopleGroup name="clerksManager">
9694     <htd:documentation xml:lang="en-US">
9695       The manager of the clerk whose user ID is passed as parameter.
9696     </htd:documentation>
9697     <htd:parameter name="clerkUserID" type="xsd:string" />
9698   </htd:logicalPeopleGroup>
9699
9700   <htd:logicalPeopleGroup name="directorClaims">
9701     <htd:documentation xml:lang="en-US">
9702       The functional director responsible for claims processing.
9703     </htd:documentation>
9704   </htd:logicalPeopleGroup>
9705
9706 </htd:logicalPeopleGroups>
9707
9708 <htd:tasks>
9709
9710   <htd:task name="ApproveClaim">
9711     <htd:documentation xml:lang="en-US">
9712       This task is used to handle claims that require manual
9713       approval.
9714     </htd:documentation>
9715
9716     <htd:interface portType="cl:ClaimsHandlingPT"
9717       operation="approve"
9718       responsePortType="cl:ClaimsHandlingCallbackPT"
9719       responseOperation="approvalResponse" />
9720
9721     <htd:priority>
9722       htd:getInput("ClaimApprovalRequest")/prio
9723     </htd:priority>
9724
9725     <htd:peopleAssignments>
9726       <htd:potentialOwners>
9727         <htd:from logicalPeopleGroup="regionalClerks">
9728           <htd:argument name="region">
9729             htd:getInput("ClaimApprovalRequest")/region
9730           </htd:argument>
9731         </htd:from>
9732       </htd:potentialOwners>
9733
9734       <htd:businessAdministrators>

```

```

9735     <htd:from logicalPeopleGroup="regionalManager">
9736         <htd:argument name="region">
9737             htd:getInput("ClaimApprovalRequest")/region
9738         </htd:argument>
9739     </htd:from>
9740 </htd:businessAdministrators>
9741 </htd:peopleAssignments>
9742
9743 <htd:delegation potentialDelegates="nobody" />
9744
9745 <htd:presentationElements>
9746
9747     <htd:name xml:lang="en-US">Approve Claim</htd:name>
9748     <htd:name xml:lang="de-DE">
9749         Genehmigung der Schadensforderung
9750     </htd:name>
9751
9752 <htd:presentationParameters>
9753     <htd:presentationParameter name="firstname"
9754         type="xsd:string">
9755         htd:getInput("ClaimApprovalRequest")/cust/firstname
9756     </htd:presentationParameter>
9757     <htd:presentationParameter name="lastname"
9758         type="xsd:string">
9759         htd:getInput("ClaimApprovalRequest")/cust/lastname
9760     </htd:presentationParameter>
9761     <htd:presentationParameter name="euroAmount"
9762         type="xsd:double">
9763         htd:getInput("ClaimApprovalRequest")/amount
9764     </htd:presentationParameter>
9765 </htd:presentationParameters>
9766
9767 <htd:subject xml:lang="en-US">
9768     Approve the insurance claim for €$euroAmount$ on behalf of
9769     $firstname$ $lastname$
9770 </htd:subject>
9771 <htd:subject xml:lang="de-DE">
9772     Genehmigung der Schadensforderung über €$euroAmount$ für
9773     $firstname$ $lastname$
9774 </htd:subject>
9775
9776 <htd:description xml:lang="en-US" contentType="text/plain">
9777     Approve this claim following corporate guideline
9778     #4711.0815/7 ...
9779 </htd:description>
9780 <htd:description xml:lang="en-US" contentType="text/html">
9781     <![CDATA[
9782     <p>
9783         Approve this claim following corporate guideline
9784         <b>#4711.0815/7</b>
9785         ...
9786     </p>
9787     ]]>
9788 </htd:description>
9789 <htd:description xml:lang="de-DE" contentType="text/plain">
9790     Genehmigen Sie diese Schadensforderung entsprechend
9791     Richtlinie Nr. 4711.0815/7 ...
9792 </htd:description>

```

```

9793 <htd:description xml:lang="de-DE" contentType="text/html">
9794 <![CDATA[
9795 <p>
9796 Genehmigen Sie diese Schadensforderung entsprechend
9797 Richtlinie
9798 <b>Nr. 4711.0815/7</b>
9799 ...
9800 </p>
9801 ]]>
9802 </htd:description>
9803
9804 </htd:presentationElements>
9805
9806
9807 <htd:deadlines>
9808
9809 <htd:startDeadline name="sendReminder">
9810 <htd:documentation xml:lang="en-US">
9811 If not started within 3 days, - escalation notifications
9812 are sent if the claimed amount is less than 10000 - to the
9813 task's potential owners to remind them or their todo - to
9814 the regional manager, if this approval is of high priority
9815 (0,1, or 2) - the task is reassigned to Alan if the
9816 claimed amount is greater than or equal 10000
9817 </htd:documentation>
9818 <htd:for>P3D</htd:for>
9819
9820 <htd:escalation name="reminder">
9821
9822 <htd:condition>
9823 <![CDATA[
9824 htd:getInput("ClaimApprovalRequest")/amount < 10000
9825 ]]>
9826 </htd:condition>
9827
9828 <htd:toParts>
9829 <htd:toPart name="firstname">
9830 htd:getInput("ClaimApprovalRequest","ApproveClaim")
9831 /firstname
9832 </htd:toPart>
9833 <htd:toPart name="lastname">
9834 htd:getInput("ClaimApprovalRequest","ApproveClaim")
9835 /lastname
9836 </htd:toPart>
9837 </htd:toParts>
9838
9839 <htd:localNotification
9840 reference="tns:ClaimApprovalReminder">
9841
9842 <htd:documentation xml:lang="en-US">
9843 Reuse the predefined notification
9844 "ClaimApprovalReminder". Overwrite the recipients with
9845 the task's potential owners.
9846 </htd:documentation>
9847
9848 <htd:peopleAssignments>
9849 <htd:recipients>
9850 <htd:from>

```



```

9851         htd:getPotentialOwners("ApproveClaim")
9852     </htd:from>
9853 </htd:recipients>
9854 </htd:peopleAssignments>
9855
9856 </htd:localNotification>
9857
9858 </htd:escalation>
9859
9860 <htd:escalation name="highPrio">
9861
9862     <htd:condition>
9863         <![CDATA[
9864             (htd:getInput("ClaimApprovalRequest")/amount < 10000
9865             && htd:getInput("ClaimApprovalRequest")/prio <= 2)
9866         ]]>
9867     </htd:condition>
9868
9869     <!-- task input implicitly passed to the notification -->
9870
9871     <htd:notification name="ClaimApprovalOverdue">
9872         <htd:documentation xml:lang="en-US">
9873             An inline defined notification using the approval data
9874             as its input.
9875         </htd:documentation>
9876
9877         <htd:interface portType="cl:ClaimsHandlingPT"
9878             operation="escalate" />
9879
9880         <htd:peopleAssignments>
9881             <htd:recipients>
9882                 <htd:from logicalPeopleGroup="regionalManager">
9883                     <htd:argument name="region">
9884                         htd:getInput("ClaimApprovalRequest")/region
9885                     </htd:argument>
9886                 </htd:from>
9887             </htd:recipients>
9888         </htd:peopleAssignments>
9889
9890         <htd:presentationElements>
9891             <htd:name xml:lang="en-US">
9892                 Claim approval overdue
9893             </htd:name>
9894             <htd:name xml:lang="de-DE">
9895                 Überfällige Schadensforderungsgenehmigung
9896             </htd:name>
9897         </htd:presentationElements>
9898
9899     </htd:notification>
9900
9901 </htd:escalation>
9902
9903 <htd:escalation name="highAmountReassign">
9904
9905     <htd:condition>
9906         <![CDATA[
9907             htd:getInput("ClaimApprovalRequest")/amount >= 10000
9908         ]]>

```

```

9909     </htd:condition>
9910
9911     <htd:reassignment>
9912         <htd:documentation>
9913             Reassign task to Alan if amount is greater than or
9914             equal 10000.
9915         </htd:documentation>
9916
9917         <htd:potentialOwners>
9918             <htd:from>
9919                 <htd:literal>
9920                     <htt:organizationalEntity>
9921                         <htt:user>Alan</htt:user>
9922                     </htt:organizationalEntity>
9923                 </htd:literal>
9924             </htd:from>
9925         </htd:potentialOwners>
9926
9927     </htd:reassignment>
9928
9929 </htd:escalation>
9930
9931 </htd:startDeadline>
9932
9933
9934 <htd:completionDeadline name="notifyManager">
9935     <htd:documentation xml:lang="en-US">
9936         When not completed within 3 hours after having been
9937         claimed, the manager of the clerk who claimed the activity
9938         is notified.
9939     </htd:documentation>
9940     <htd:for>PT3H</htd:for>
9941
9942     <htd:escalation name="delayedApproval">
9943
9944         <htd:notification name="ClaimApprovalOverdue">
9945             <htd:documentation xml:lang="en-US">
9946                 An inline defined notification using the approval data
9947                 as its input.
9948             </htd:documentation>
9949
9950             <htd:interface portType="cl:ClaimsHandlingPT"
9951                 operation="escalate" />
9952
9953             <htd:peopleAssignments>
9954                 <htd:recipients>
9955                     <htd:from logicalPeopleGroup="clerksManager">
9956                         <htd:argument name="clerkUserID">
9957                             htd:getActualOwner("ApproveClaim")
9958                         </htd:argument>
9959                     </htd:from>
9960                 </htd:recipients>
9961             </htd:peopleAssignments>
9962
9963             <htd:presentationElements>
9964                 <htd:name xml:lang="en-US">
9965                     Claim approval overdue
9966                 </htd:name>

```

```

9967         <htd:name xml:lang="de-DE" >
9968             Überfällige Schadensforderungsgenehmigung
9969         </htd:name>
9970     </htd:presentationElements>
9971
9972     </htd:notification>
9973
9974     </htd:escalation>
9975 </htd:completionDeadline>
9976
9977 <htd:completionDeadline name="notifyDirector">
9978     <htd:documentation xml:lang="en-US" >
9979         When not completed within 2 days after having been
9980         claimed, the functional director of claims processing is
9981         notified.
9982     </htd:documentation>
9983     <htd:for>P2D</htd:for>
9984
9985     <htd:escalation name="severelyDelayedApproval">
9986
9987     <htd:notification name="ClaimApprovalOverdue">
9988         <htd:documentation xml:lang="en-US" >
9989             An inline defined notification using the approval data
9990             as its input.
9991         </htd:documentation>
9992
9993         <htd:interface portType="cl:ClaimsHandlingPT"
9994             operation="escalate" />
9995
9996         <htd:peopleAssignments>
9997             <htd:recipients>
9998                 <htd:from logicalPeopleGroup="directorClaims">
9999                     <htd:argument name="clerkUserID">
10000                         htd:getActualOwner("ApproveClaim")
10001                     </htd:argument>
10002                 </htd:from>
10003             </htd:recipients>
10004         </htd:peopleAssignments>
10005
10006     </htd:notification>
10007     <htd:presentationElements>
10008         <htd:name xml:lang="en-US" >
10009             Claim approval severely overdue
10010         </htd:name>
10011         <htd:name xml:lang="de-DE" >
10012             Hochgradig überfällige Schadensforderungsgenehmigung
10013         </htd:name>
10014     </htd:presentationElements>
10015
10016     </htd:notification>
10017
10018     </htd:escalation>
10019 </htd:completionDeadline>
10020
10021 </htd:deadlines>
10022
10023 </htd:task>
10024
10025 </htd:tasks>

```

```

10025
10026 <htd:notifications>
10027
10028 <htd:notification name="ClaimApprovalReminder">
10029 <htd:documentation xml:lang="en-US">
10030 This notification is used to remind people of pending
10031 out-dated claim approvals. Recipients of this notification
10032 maybe overridden when it is referenced.
10033 </htd:documentation>
10034
10035 <htd:interface portType="cl:ClaimApprovalReminderPT"
10036 operation="notify" />
10037
10038 <htd:peopleAssignments>
10039 <htd:recipients>
10040 <htd:from>
10041 <htd:literal>
10042 <htt:organizationalEntity>
10043 <htt:user>Alan</htt:user>
10044 <htt:user>Dieter</htt:user>
10045 <htt:user>Frank</htt:user>
10046 <htt:user>Gerhard</htt:user>
10047 <htt:user>Ivana</htt:user>
10048 <htt:user>Karsten</htt:user>
10049 <htt:user>Matthias</htt:user>
10050 <htt:user>Patrick</htt:user>
10051 </htt:organizationalEntity>
10052 </htd:literal>
10053 </htd:from>
10054 </htd:recipients>
10055 </htd:peopleAssignments>
10056
10057 <htd:presentationElements>
10058
10059 <htd:name xml:lang="en-US">Approve Claim</htd:name>
10060 <htd:name xml:lang="de-DE">
10061 Genehmigung der Schadensforderung
10062 </htd:name>
10063
10064 <htd:presentationParameters>
10065 <htd:presentationParameter name="firstname"
10066 type="xsd:string">
10067 htd:getInput("firstname")
10068 </htd:presentationParameter>
10069 <htd:presentationParameter name="lastname"
10070 type="xsd:string">
10071 htd:getInput("lastname")
10072 </htd:presentationParameter>
10073 <htd:presentationParameter name="id" type="xsd:string">
10074 htd:getInput("taskId")
10075 </htd:presentationParameter>
10076 </htd:presentationParameters>
10077
10078 <htd:subject xml:lang="en-US">
10079 Claim approval for $firstname$, $lastname$ is overdue. See
10080 task $id$.
10081 </htd:subject>
10082

```

```
10083     </htd:presentationElements>
10084
10085     </htd:notification>
10086
10087     </htd:notifications>
10088
10089 </htd:humanInteractions>
```

10090 J. Acknowledgements

10091 The following individuals have participated in the creation of this specification and are gratefully
10092 acknowledged:

10093

10094 **Members of the BPEL4People Technical Committee:**

10095 Phillip Allen, Microsoft Corporation

10096 Ashish Agrawal, Adobe Systems

10097 Mike Amend, BEA Systems, Inc.

10098 Stefan Baeuerle, SAP AG

10099 Charlton Barreto, Adobe Systems

10100 Justin Brunt, TIBCO Software Inc.

10101 Martin Chapman, Oracle Corporation

10102 Luc Clément, Active Endpoints, Inc.

10103 Manoj Das, Oracle Corporation

10104 Alireza Farhoush, TIBCO Software Inc.

10105 Mark Ford, Active Endpoints, Inc.

10106 Sabine Holz, SAP AG

10107 Dave Ings, IBM

10108 Gershon Janssen, Individual

10109 Diane Jordan, IBM

10110 Anish Karmarkar, Oracle Corporation

10111 Ulrich Keil, SAP AG

10112 Oliver Kieselbach, SAP AG

10113 Matthias Kloppmann, IBM

10114 Dieter König, IBM

10115 Marita Kruempelmann, SAP AG

10116 Frank Leymann, IBM

10117 Mark Little, Red Hat

10118 Alexander Malek, Microsoft Corporation

10119 Ashok Malhotra, Oracle Corporation

10120 Mike Marin, IBM

10121 Vinkesh Mehta, Deloitte Consulting LLP

10122 Jeff Mischkinsky, Oracle Corporation

10123 Ralf Mueller, Oracle Corporation

10124 Krasimir Nedkov, SAP AG

10125 Benjamin Notheis, SAP AG

10126 Michael Pellegrini, Active Endpoints, Inc.

10127 Hannah Petereit, SAP AG

10128 Gerhard Pfau, IBM

10129 Karsten Ploesser, SAP AG

10130 Ravi Rangaswamy, Oracle Corporation
10131 Alan Rickayzen, SAP AG
10132 Michael Rowley, BEA Systems, Inc.
10133 Ron Ten-Hove, Sun Microsystems
10134 Ivana Trickovic, SAP AG
10135 Alessandro Triglia, OSS Nokalva
10136 Claus von Riegen, SAP AG
10137 Peter Walker, Sun Microsystems
10138 Franz Weber, SAP AG
10139 Prasad Yendluri, Software AG, Inc.

10140

10141 **WS-HumanTask 1.0 Specification Contributors:**

10142 Ashish Agrawal, Adobe
10143 Mike Amend, BEA
10144 Manoj Das, Oracle
10145 Mark Ford, Active Endpoints
10146 Chris Keller, Active Endpoints
10147 Matthias Kloppmann, IBM
10148 Dieter König, IBM
10149 Frank Leymann, IBM
10150 Ralf Müller, Oracle
10151 Gerhard Pfau, IBM
10152 Karsten Plösser, SAP
10153 Ravi Rangaswamy, Oracle
10154 Alan Rickayzen, SAP
10155 Michael Rowley, BEA
10156 Patrick Schmidt, SAP
10157 Ivana Trickovic, SAP
10158 Alex Yiu, Oracle
10159 Matthias Zeller, Adobe

10160

10161 The following individuals have provided valuable input into the design of this specification: Dave Ings,
10162 Diane Jordan, Mohan Kamath, Ulrich Keil, Matthias Kruse, Kurt Lind, Jeff Mischkinsky, Bhagat Nainani,
10163 Michael Pellegrini, Lars Rueter, Frank Ryan, David Shaffer, Will Stallard, Cyrille Waguët, Franz Weber,
10164 and Eric Wittmann.

10166

L. Revision History

10167 [optional; should not be included in OASIS Standards]

10168

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
CD-06-rev1	2010-02-20	Dieter König	Issue 133, 134, 135, 136, 137, 139, 140, 141, 142, 143 Editorial: -- Sorted several operation lists/tables (API operations and XPath functions) -- Copied modified XML artifacts back to appendix
CD-07	2010-03-03	Luc Clément	Creating of CD07, Copyright date updates and cover page annotation as Public Review 02

10169