



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

Committee Draft 10 / Public Review Draft 04

23 June 2010

Specification URIs:

This Version:

<http://docs.oasis-open.org/bpel4people/ws-humantask-1.1-spec-cd-10.html>
<http://docs.oasis-open.org/bpel4people/ws-humantask-1.1-spec-cd-10.doc> (Authoritative format)
<http://docs.oasis-open.org/bpel4people/ws-humantask-1.1-spec-cd-10.pdf>

Previous Version:

<http://docs.oasis-open.org/bpel4people/ws-humantask-1.1-spec-cd-09.html>
<http://docs.oasis-open.org/bpel4people/ws-humantask-1.1-spec-cd-09.doc>
<http://docs.oasis-open.org/bpel4people/ws-humantask-1.1-spec-cd-09.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® 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 name "OASIS" is a trademark 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	33
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	47
4.9	Elements for Handling Timeouts and Escalations.....	51
4.10	Human Task Behavior and State Transitions.....	58
4.10.1	Normal processing of a Human Task	58
4.10.2	Releasing a Human Task	59
4.10.3	Delegating or Forwarding a Human Task.....	59
4.10.4	Sub Task Event Propagation	59
4.11	History of a Human Task	60
4.11.1	Task Event Types and Data	61
4.11.2	Retrieving the History	63
5	Lean Tasks	66
5.1	Overall Syntax	66
5.2	Properties	66
5.3	Message Schema.....	66
5.4	Example: ToDoTask.....	68
6	Notifications	69
6.1	Overall Syntax	69
6.2	Properties	70
6.3	Notification Behavior and State Transitions	70
7	Programming Interfaces.....	71
7.1	Operations for Client Applications	71
7.1.1	Participant Operations	71
7.1.2	Simple Query Operations	83
7.1.3	Advanced Query Operation	86
7.1.4	Administrative Operations.....	89
7.1.5	Operation Authorizations	90
7.2	XPath Extension Functions	92
8	Interoperable Protocol for Advanced Interaction with Human Tasks	99
8.1	Human Task Coordination Protocol Messages.....	101
8.2	Protocol Messages	102
8.2.1	Protocol Messages Received by a Task Parent.....	102
8.2.2	Protocol Messages Received by a Task	102
8.3	WSDL of the Protocol Endpoints.....	102
8.3.1	Protocol Endpoint of the Task Parent.....	102
8.3.2	Protocol Endpoint of the Task.....	103
8.4	Providing Human Task Context.....	103
8.4.1	SOAP Binding of Human Task Context.....	103

8.4.2	Overriding Task Definition People Assignments	104
8.5	Human Task Policy Assertion	105
9	Task Parent Interactions with Lean Tasks.....	106
9.1	Operations for Task Parent Applications.....	106
9.2	Lean Task Interactions	106
9.2.1	Register a Lean Task Definition.....	106
9.2.2	Unregister a Lean Task Definition	107
9.2.3	List Lean Task Definitions.....	107
9.2.4	Create a Lean Task	108
9.2.5	Endpoints for Lean Task Operations	109
10	Providing Callback Information for Human Tasks.....	111
10.1	EPR Information Model Extension	111
10.2	XML Infoset Representation.....	111
10.3	Message Addressing Properties	113
10.4	SOAP Binding.....	114
11	Security Considerations	117
12	Conformance.....	118
A.	Portability and Interoperability Considerations	119
B.	WS-HumanTask Language Schema	120
C.	WS-HumanTask Data Types Schema.....	135
D.	WS-HumanTask Client API Port Type.....	144
E.	WS-HumanTask Parent API Port Type.....	188
F.	WS-HumanTask Protocol Handler Port Types	194
G.	WS-HumanTask Context Schema	196
H.	WS-HumanTask Policy Assertion Schema.....	199
I.	Sample.....	200
J.	Acknowledgements.....	210
K.	Revision History	212

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.ietf.org/rfc/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.ietf.org/rfc/rfc2396.txt>

59 [RFC 3066]

60 Tags for the Identification of Languages, H. Alvestrand, IETF, January 2001, available via
61 <http://www.ietf.org/rfc/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 Recommendation 04 September 2007, available via
79 <http://www.w3.org/TR/ws-policy/>

80 [WS-PolAtt]

81 Web Services Policy 1.5 - Attachment, W3C Recommendation 04 September 2007, available via
82 <http://www.w3.org/TR/ws-policy-attach/>

83 [XML Infoset]

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

86 [XML Namespaces]

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

89 [XML Schema Part 1]

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

92 **[XML Schema Part 2]**

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

95 **[XMLSpec]**

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

98 **[XPath 1.0]**

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

101 **1.3 Non-Normative References**

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

103 **1.4 Conformance Targets**

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

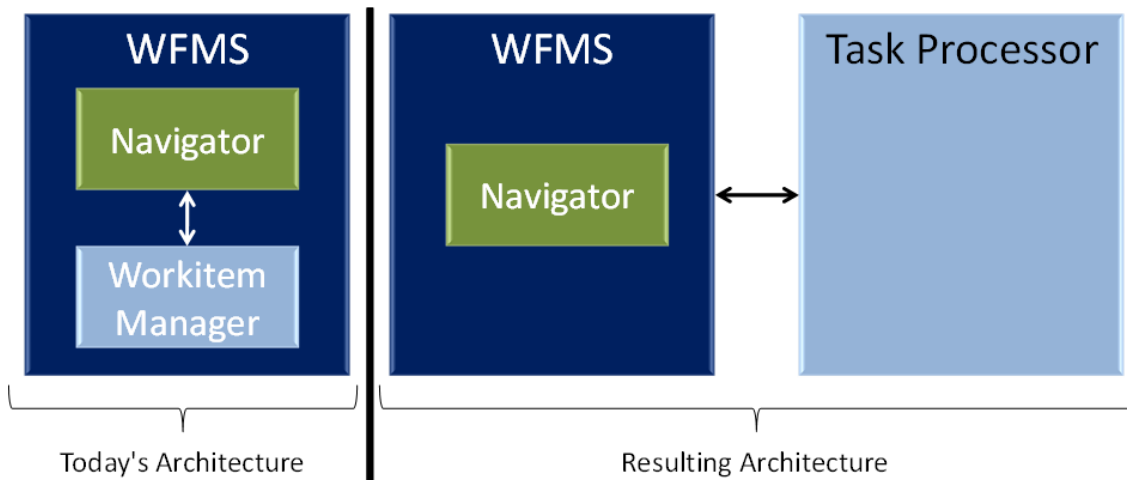
- 105 • WS-HumanTask Definition
106 A WS-HumanTask Definition is any artifact that complies with the human interaction schema and
107 additional constraints defined in this document.
- 108 • WS-HumanTask Processor
109 A WS-HumanTask Processor is any implementation that accepts a WS-HumanTask definition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

117

119 1.5 Overall Architecture

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

127



128

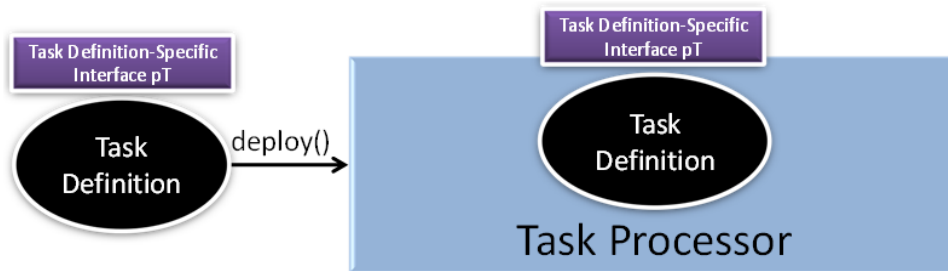
129 **Figure 1- Architectural Impact of WS-HumanTask on Workflow Management Systems**

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

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

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

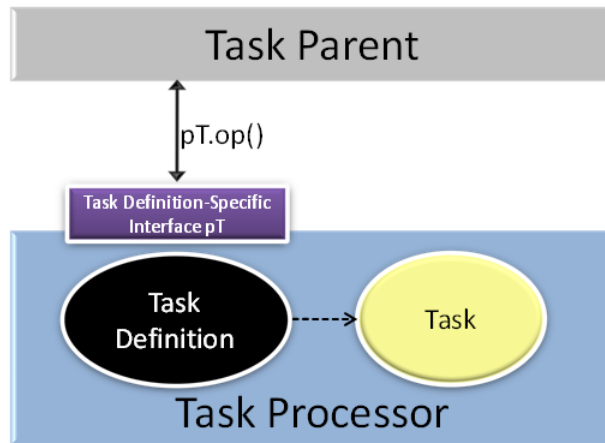
147



148
149
150
151
152
153
154
155

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.



156
157

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.

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.

175
176

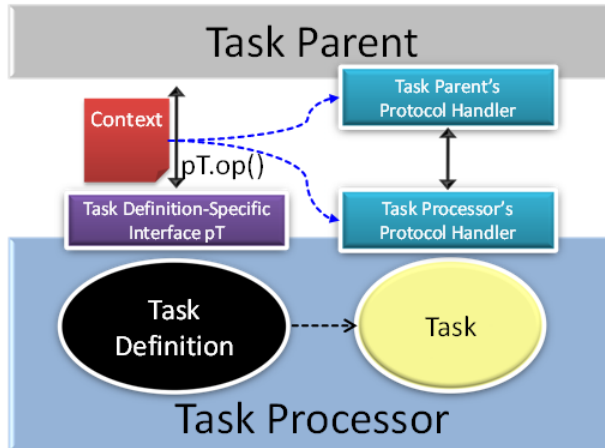


Figure 4 - Establishing a Protocol Channel

177
178
179
180
181
182
183
184
185
186

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

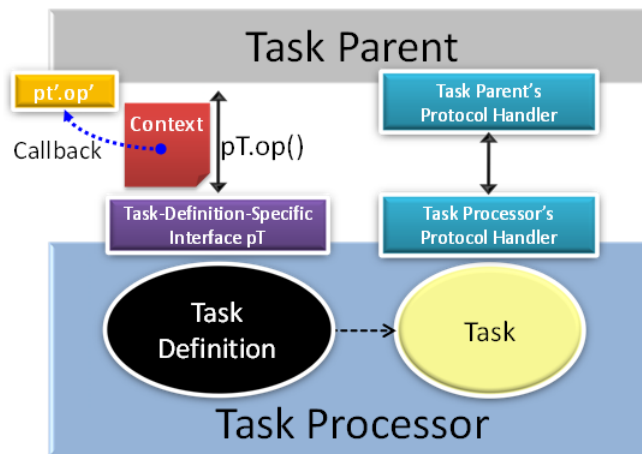


Figure 5 - Passing Callback Information for Long Running Tasks

187
188
189
190
191
192
193
194
195
196
197
198
199
200
201

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

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

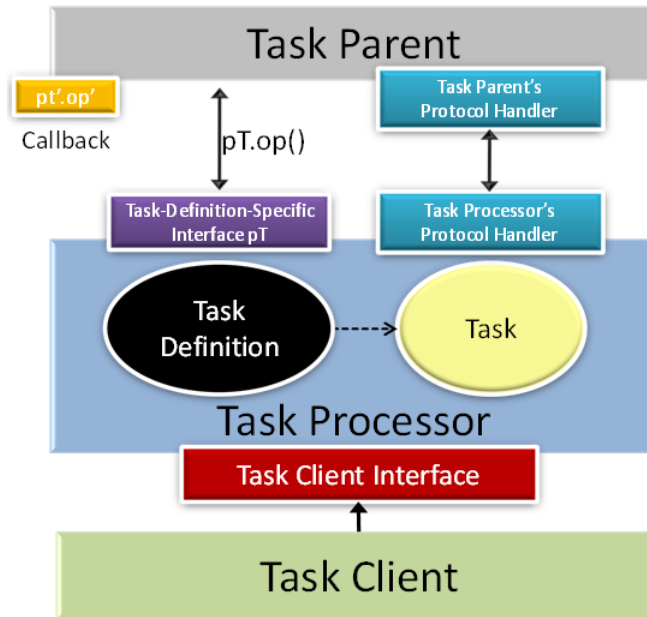
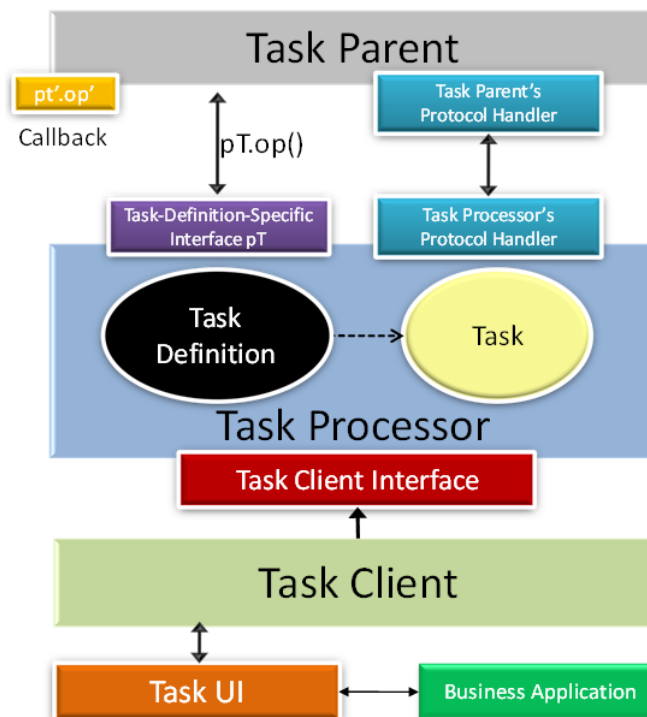


Figure 6 - Task List Client and Corresponding Interface

202
203
204
205
206
207
208
209
210
211
212
213

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.

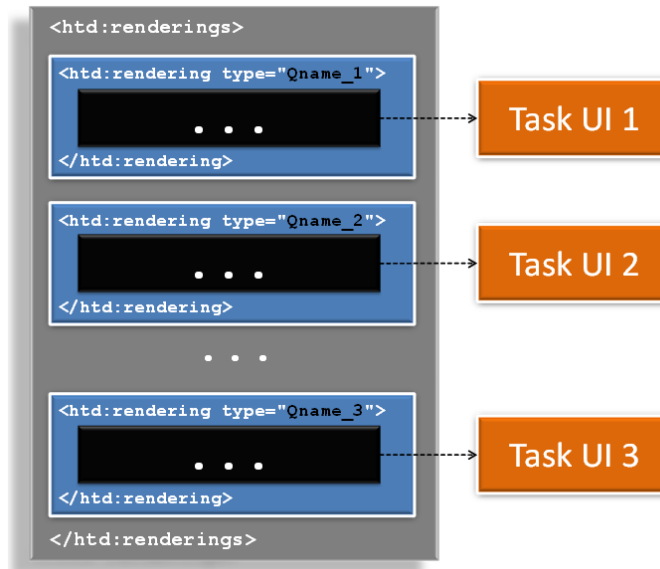


214

215

Figure 7 - Overall Architecture of a Human Task Infrastructure

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

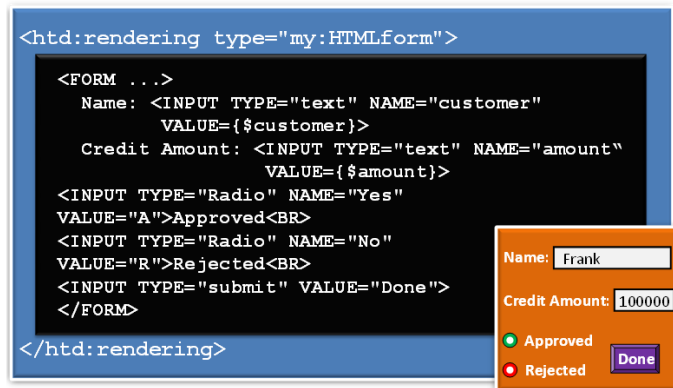


222

223

Figure 8 - Potential Renderings of a Task

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



229

230

Figure 9 - Sample Rendering of a Task

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

234 2 Language Design

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

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

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

247 2.1 Dependencies on Other Specifications

248 WS-HumanTask utilizes the following specifications:

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

255 2.1.1 Namespaces Referenced

256 WS-HumanTask references these namespaces:

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

261 2.2 Language Extensibility

262 The WS-HumanTask extensibility mechanism allows:

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

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

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

272 2.3 Overall Language Structure

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

276 2.3.1 Syntax

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

311 2.3.2 Properties

312 The <humanInteractions> element has the following properties:

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

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

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

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

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

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

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

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

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

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

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

```
398 <htd:documentation xml:lang="xsd:language">  
399   ...  
400 </htd:documentation>
```

401 2.4 Default use of XPath 1.0 as an Expression Language

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

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

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

416

3 Concepts

417

3.1 Generic Human Roles

418

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:

419

420

- Task initiator

421

- Task stakeholders

422

- Potential owners

423

- Actual owner

424

- Excluded owners

425

- Business administrators

426

- Notification recipients

427

428

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.

429

430

431

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.

432

433

434

435

436

437

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.

438

439

440

441

442

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.

443

444

445

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.

446

447

448

449

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.

450

451

452

453

454

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.

455

456

457

458

459 **3.2 Composite Tasks and Sub Tasks**

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

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

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

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

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

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

474 **3.2.1 Composite Tasks by Definition**

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

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

490 **3.2.2 Composite Tasks Created Adhoc at Runtime**

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

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

496 **3.3 Routing Patterns**

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

502 3.4 Relationship of Composite Tasks and Routing Patterns

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

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

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

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

515 3.5 Assigning People

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

- 518 • Via logical people groups (see 3.5.1 "Using Logical People Groups")
- 519 • Via literals (see 3.5.2 "Using Literals")
- 520 • Via expressions e.g., by retrieving data from the input message of the human task (see 3.5.3
521 "Using Expressions").
- 522 • In a well-defined order using Routing Patterns (see 4.7.1 "Routing Patterns")

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

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

528 Syntax:

```
529 <htd:peopleAssignments>  
530  
531   <htd:genericHumanRole>+  
532     <htd:from>...</htd:from>  
533   </htd:genericHumanRole>  
534  
535   <htd:potentialOwners>+  
536     fromPattern+  
537   </htd: potentialOwners>  
538  
539 </htd:peopleAssignments>
```

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

```
542 <htd:excludedOwners>  
543   <htd:from>...</htd:from>  
544 </htd:excludedOwners>  
545  
546 <htd:taskInitiator>  
547   <htd:from>...</htd:from>  
548 </htd:taskInitiator>
```

```
549
550 <htd:taskStakeholders>
551   <htd:from>...</htd:from>
552 </htd:taskStakeholders>
553
554 <htd:businessAdministrators>
555   <htd:from>...</htd:from>
556 </htd:businessAdministrators>
557
558 <htd:recipients>
559   <htd:from>...</htd:from>
560 </htd:recipients>
```

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

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

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

579 3.5.1 Using Logical People Groups

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

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

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

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

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

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

619 **Syntax:**

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

625
626 The `logicalPeopleGroup` attribute refers to a `logicalPeopleGroup` definition. The element
627 `<argument>` is used to pass values used in the people query. The `expressionLanguage` attribute
628 specifies the language used in the expression. The attribute is optional. If not specified, the default
629 language as inherited from the closest enclosing element that specifies the attribute MUST be used by
630 WS-HumanTask Processor.

631 **Example:**

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

639 **3.5.2 Using Literals**

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

643

644

645 **Syntax:**

```
646 <htd:from>
647   <htd:literal>
648     ... literal value ...
649   </htd:literal>
650 </htd:from>
```

651 **Example specifying user identifiers:**

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

668 **Example specifying group names:**

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

678 **3.5.3 Using Expressions**

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

682 **Syntax:**

```
683 <htd:from expressionLanguage="anyURI"?>
684   expression
685 </htd:from>
```

686

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

690

691

692 **Example:**

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

703 **3.5.4 Data Type for Organizational Entities**

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

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

725 **3.5.5 Subtasks**

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

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

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

745 3.6 Task Rendering

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

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

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

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

763 Versatility of deployment determines which software within a particular constellation performs the
764 presentation rendering.

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

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

775 3.7 Lean Tasks

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

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

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

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

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

802 **3.8 Task Instance Data**

803 Task instance data falls into three categories:

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

810 **3.8.1 Presentation Data**

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

814 **3.8.2 Context Data**

815 The task context includes the following:

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

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

825 **3.8.3 Operational Data**

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

829 3.8.3.1 Ad-hoc Attachments

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

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

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

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

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

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

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

851 The `attachedBy` element indicates who added the attachment. It is a single user (type `htt:tUser`).

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

855 Attachment Info Data Type

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

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

871 Attachment Data Type

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

```
873 <xsd:element name="attachment" type="tAttachment" />  
874 <xsd:complexType name="tAttachment">  
875   <xsd:sequence>  
876     <xsd:element ref="attachmentInfo" />  
877     <xsd:element name="value" type="xsd:anyType" />
```

```
878 </xsd:sequence>
879 </xsd:complexType>
```

880 3.8.3.2 Comments

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

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

888 The `addedBy` element indicates who added the comment. It is a single user (type `htt:tUser`).

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

890 The `lastModifiedBy` element indicates who last modified the comment. It is a single user (type
891 `htt:tUser`).

892 Comment Data Type

893 The following data type is used to return comments.

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

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

908 3.8.4 Data Types for Task Instance Data

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

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

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

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

922

923

924 TaskAbstract Data Type

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

978

979

980 **TaskDetails Data Type**

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

```

1038     <xsd:element name="hasComments"
1039               type="xsd:boolean" minOccurs="0"/>
1040     <xsd:element name="escalated"
1041               type="xsd:boolean" minOccurs="0"/>
1042     <xsd:element name="searchBy"
1043               type="xsd:string" minOccurs="0"/>
1044     <xsd:element name="outcome"
1045               type="xsd:string" minOccurs="0"/>
1046     <xsd:element name="parentTaskId"
1047               type="xsd:anyURI" minOccurs="0"/>
1048     <xsd:element name="hasSubTasks"
1049               type="xsd:boolean" minOccurs="0"/>
1050     <xsd:any namespace="##other" processContents="lax"
1051            minOccurs="0" maxOccurs="unbounded"/>
1052   </xsd:sequence>
1053 </xsd:complexType>

```

1054 Common Data Types

```

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

```


1095 `</xsd:simpleType>`

1096 **3.8.5 Sub Tasks**

1097 To support sub tasks the task instance data gets enhanced by the following (optional) parameters:

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

1103 4 Human Tasks

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

1106 4.1 Overall Syntax

1107 Definition of human tasks:

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

```

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

```

1171 4.2 Properties

1172 The following attributes and elements are defined for tasks:

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

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

1240 4.3 Presentation Elements

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

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

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

1251 Human readable information can be specified in multiple languages.

1252 **Syntax:**

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

1277 **Properties**

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

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

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

1308 **Example:**

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

1358
1359 `</htd:presentationElements>`

1360 4.4 Task Possible Outcomes

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

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

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

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

1382 4.5 Elements for Rendering Tasks

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

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

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

1396

1397

1398 **Syntax:**

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

1404 **4.6 Elements for Composite Tasks**

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

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

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

1432 **Syntax:**

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



```

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

```

1455 *Standard-overriding-elements* is used in the syntax above as a shortened form of the following list of
1456 elements:

```

1457 <htd:priority expressionLanguage="anyURI"? >
1458     integer-expression
1459 </htd:priority>
1460
1461 <htd:peopleAssignments>?
1462     <htd:genericHumanRole>
1463         <htd:from>...</htd:from>
1464     </htd:genericHumanRole>
1465 </htd:peopleAssignments>

```

1466 4.7 Elements for People Assignment

1467 The <peopleAssignments> element is used to assign people to a task. For each generic human role, a
1468 people assignment element can be specified. A WS-HumanTask Definition MUST specify a people
1469 assignment for potential owners of a human task. An empty <potentialOwners> element is used to
1470 specify that no potential owner is assigned by the human task's definition but another means is used e.g.
1471 nomination. Specifying people assignments for task stakeholders, task initiators, excluded owners and
1472 business administrators is optional. Human tasks never specify recipients. A WS-HumanTask Definition
1473 MUST NOT specify people assignments for actual owners.

1474 **Syntax:**

```

1475 <htd:peopleAssignments>
1476     <htd:potentialOwners>
1477         ...
1478     </htd:potentialOwners>
1479     <htd:excludedOwners>?
1480         ...
1481     </htd:excludedOwners>
1482     <htd:taskInitiator>?
1483         ...
1484     </htd:taskInitiator>
1485     <htd:taskStakeholders>?
1486         ...
1487     </htd:taskStakeholders>
1488     <htd:businessAdministrators>?
1489         ...
1490     </htd:businessAdministrators>
1491 </htd:peopleAssignments>

```

1492 People assignments can result in a set of values or an empty set. In case people assignment results in an
1493 empty set then the task potentially requires administrative attention. This is out of scope of the
1494 specification, except for people assignments for potential owners (see section 4.10.1 "Normal processing
1495 of a Human Task" for more details).

1496
1497

1498 **Example:**

```
1499 <htd:peopleAssignments>
1500   <htd:potentialOwners>
1501     <htd:from logicalPeopleGroup="regionalClerks">
1502       <htd:argument name="region">
1503         htd:getInput("ClaimApprovalRequest")/region
1504       </htd:argument>
1505     </htd:from>
1506   </htd:potentialOwners>
1507
1508   <htd:businessAdministrators>
1509     <htd:from logicalPeopleGroup="regionalManager">
1510       <htd:argument name="region">
1511         htd:getInput("ClaimApprovalRequest")/region
1512       </htd:argument>
1513     </htd:from>
1514   </htd:businessAdministrators>
1515 </htd:peopleAssignments>
```

1516 4.7.1 Routing Patterns

1517 Tasks can be assigned to people in sequence and parallel. Elements `htd:sequence` and
1518 `htd:parallel` elements in `htd:potentialOwners` are used to represent such assignments.

1519 4.7.1.1 Parallel Pattern

1520 A task can be assigned to people in parallel using the `htd:parallel` element. The `htd:parallel`
1521 element is defined as follows:

- 1522 • The `htd:from` element defines the parallel potential owners. This can evaluate to multiple
1523 users/groups.
- 1524 • The attribute 'type' in `htd:parallel` identifies how parallel assignments are created for the
1525 multiple users/groups returned from `htd:from`. If type is 'all' then an assignment MUST be
1526 created for each user returned by `htd:from`. If type is 'single' then an assignment MUST be
1527 created for each `htd:from` clause (this assignment could have n potential owners). The default
1528 value of type is 'all'.
- 1529 • The `htd:parallel` and `htd:sequence` elements define nested routing patterns within the
1530 parallel routing pattern.
- 1531 • The `htd:completionBehavior` defines when the routing pattern completes. The completion
1532 criteria also define how the result is constructed for the parent task when a parallel routing
1533 pattern is complete.

1534 Each parallel assignment MUST result in a separate sub task. Sub tasks created for each parallel
1535 assignment MUST identify the parent task using the `htd:parentTaskId`.

1536

1537

1538 **Syntax:**

```
1539 <htd:potentialOwners>  
1540   <htd:parallel type="all|single"?>  
1541     <htd:completionBehavior?>  
1542       ...  
1543     </htd:completionBehavior>  
1544     <htd:from>*</htd:from>  
1545     ...  
1546   </htd:parallel>  
1547   pattern*  
1548 </htd:potentialOwners>  
1549 </htd:potentialOwners>
```

1550 **Example:**

```
1551 <htd:peopleAssignments>  
1552   <htd:potentialOwners>  
1553     <htd:parallel type="all">  
1554       <htd:from>  
1555         htd:getInput ("ClaimApprovalRequest") /claimAgent  
1556       </htd:from>  
1557     </htd:parallel>  
1558   </htd:potentialOwners>  
1559 </htd:peopleAssignments>
```

1560 **4.7.1.2 Sequential Pattern**

1561 A task can be assigned to people in sequence using the `htd:sequence` element. The `htd:sequence`
1562 is defined as follows:

- 1563 • The `htd:from` element can evaluate to multiple users/groups.
- 1564 • The attribute 'type' in `htd:sequence` identifies how sequential assignments are created for the
1565 multiple users/groups returned from `htd:from`. If type is 'all' an assignment MUST be created
1566 for each user returned by `htd:from`. If type is 'single', an assignment MUST be created for each
1567 `htd:from` clause (this assignment could have with n potential owners). The default value of type
1568 is 'all'.
- 1569 • The `htd:parallel` and `htd:sequence` elements define nested routing patterns within the
1570 sequential routing pattern.
- 1571 • The `htd:completionBehavior` defines when the routing pattern completes. The completion
1572 criteria also define how the result is constructed for the parent task when a sequential routing
1573 pattern is complete.

1574 Sequential routing patterns MUST use a separate sub task for each step in a sequential pattern. Sub
1575 tasks created for each sequential assignment MUST identify the parent task using the
1576 `htd:parentTaskId`.

1577

1578

1579 **Syntax:**

```
1580 <htd:potentialOwners>
1581   <htd:sequence type="all|single"?>
1582     <htd:completionBehavior?>
1583       ...
1584     </htd:completionBehavior>
1585     <htd:from>*
1586     ...
1587   </htd:from>
1588   pattern*
1589 </htd:sequence>
1590 </htd:potentialOwners>
```

1591 **Example:**

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

1608 4.8 Completion Behavior

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

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

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

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

1629

```

1630 <htd:completionBehavior completionAction="manual|automatic"?>?
1631   <htd:completion name="NCName">*
1632     <htd:condition ... >
1633       ...
1634     </htd:condition>
1635     <htd:result>?
1636     ...
1637     <htd:result>
1638   </htd:completion>
1639 <htd:defaultCompletion>?
1640   <htd:result>
1641   ...
1642   <htd:result>
1643 </htd:defaultCompletion>
1644 </htd:completionBehavior>

```

1645 The `completionBehavior` element has optional attribute `completionAction`. This optional
1646 attribute specifies how the task, routing pattern, or composite task is completed. If the value is set to
1647 "manual" the task or parent task MUST be completed explicitly by the actual owner as soon as the
1648 completion conditions evaluate to true. If the value is set to "automatic" the task or parent task MUST be
1649 set to complete as soon as the completion conditions evaluate to true. For routing patterns, the
1650 `completionAction` attribute MUST have value "automatic". The default value for this attribute is
1651 "automatic".

1652 If `completionBehavior` is not specified, the default behavior is that of a `completionBehavior` with
1653 `completionCondition` is "true" and a `completionAction` of "manual" for simple and composite
1654 tasks, and "automatic" for routing patterns.

1655 4.8.1 Completion Conditions

1656 A completion condition defines when a task or a set of sub tasks associated with the parent task is
1657 considered completed. It is specified Boolean expression which can refer to input data of the task, the
1658 parent task or its sub tasks, output data produced by already finished sub tasks, or other data obtained
1659 from WS-HumanTask API calls (e.g. the number of sub tasks), or functions that test that some designated
1660 amount of time has passed.

1661 The completion condition MUST be defined using an `htd:condition` element.

```

1662 <htd:condition expressionLanguage="anyURI"?>
1663   boolean expression
1664 </htd:condition>

```

1665 Within the Boolean expression of a completion condition, aggregation functions can be used to evaluate
1666 output data produced by the already finished sub tasks of the parent task.

1667 If an error (e.g. division by zero) occurs during the condition evaluation then the condition MUST be
1668 considered to have evaluated to "false".

1669 The time functions that are available are defined as follows:

- 1670 • `boolean htd:waitFor(string)`
 - 1671 ○ The parameter is an XPath expression evaluating to a string conforming to the definition
 - 1672 ○ of the XML Schema type `duration`
 - 1673 ○ The return value is `true` after the specified duration has elapsed, otherwise `false`
- 1674 • `boolean htd:waitUntil(string)`
 - 1675 ○ The parameter is an XPath expression evaluating to a string conforming to the definition
 - 1676 ○ of the XML Schema type `dateTime`
 - 1677 ○ The return value is `true` after the specified absolute time has passed, otherwise `false`.

1678 Completion conditions of a task without subtasks MUST use only time functions.

1679 4.8.1.1 Evaluating the Completion Condition

1680 The time functions in the completion condition are be evaluated with respect to the beginning of execution
1681 of the task or parent task on which the completion is defined. To achieve this, the evaluation of the
1682 `htd:waitFor` and `htd:waitUntil` calls within the condition are treated differently from the rest of the
1683 expression. When the containing task or parent task is created, the actual parameter expression for any
1684 `htd:waitFor` and `htd:waitUntil` calls MUST be evaluated and the completion condition should be
1685 rewritten to replace these calls with only `htd:waitUntil` calls with constant parameters. The durations
1686 calculated for any `htd:waitFor` calls MUST be converted into absolute times and rewritten as
1687 `htd:waitUntil` calls. The result of these replacements is called the *preprocessed completion*
1688 *condition*.

1689

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

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

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

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

1699 Example:

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

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

1723

1724 4.8.2 Result Construction from Parallel Subtasks

1725 When multiple sub tasks are created in order let several people work on their own sub task in parallel
1726 then the outputs of these sub tasks sometimes need to be combined for the creation of the parent task's
1727 output.

1728 If all sub tasks have the same interface definition (as in routing patterns) then the result construction can
1729 be defined in a declarative way using aggregation functions. Alternatively, the result may be created using
1730 explicit assignments.

1731 The result construction **MUST** be defined as `htd:result` element, containing one or more
1732 `htd:aggregate` or `htd:copy` elements, executed in the order in which they appear in the task
1733 definition.

```
1734 <htd:result>  
1735   (  
1736     <htd:aggregate ... />  
1737     |  
1738     <htd:copy> ... </htd:copy>  
1739   )+  
1740 </htd:result>
```

1741 4.8.2.1 Declarative Result Aggregation

1742 An `htd:aggregate` element describes the result aggregation for a leaf element of the parent task's
1743 output document. In most cases, this approach is only meaningful for routing patterns with identical sub
1744 task interfaces. Note that the construction of (complex-typed) non-leaf elements is out of scope of the
1745 declarative result aggregation.

```
1746 <htd:aggregate part="NCName"?  
1747               location="query"?  
1748               condition="bool-expr"?  
1749               function="function-call"/>+
```

1750 The `htd:aggregate` element is defined as follows:

- 1751 • The optional `part` attribute **MUST** contain the name of a WSDL part. The `part` attribute **MUST** be
1752 specified when the task interface is defined using a WSDL message with more than one WSDL
1753 part.
- 1754 • The optional `location` attribute **MUST** contain a query pointing to the location of a leaf element
1755 of the tasks' output documents:
 - 1756 ○ For each parallel sub task, this is the location of exactly one element of the sub task's
1757 output document that is processed by the aggregation function. Each sub tasks' output
1758 element is (conditionally) added to a node-set passed as parameter to the aggregation
1759 function.
 - 1760 ○ For the parent task, this is the element created in the task's output document that is the
1761 computed return value of the aggregation function.
- 1762 • The optional `condition` attribute **MUST** contain a Boolean expression evaluated on each sub
1763 task's output document. If the expression evaluates to `true` then the sub task's output element
1764 identified by `location` is added to the node-set passed to the aggregation function.
- 1765 • The mandatory `function` attribute contains the name of the aggregation function (QName; see
1766 a list of supported aggregation functions in section 7.2) and optional arguments, in the following
1767 form:
1768 `FunctionName '(' (Argument (',' Argument)*)? ')'`
1769 **Important:**
 - 1770 ○ The first parameter of each aggregation function is the node-set of sub task's output
1771 elements to be aggregated. This parameter is inserted implicitly and **MUST NOT** be
1772 specified within the `function` attribute.

- 1773 o Within the `function` attribute, function arguments MUST be specified only for *additional*
1774 parameters defined for an aggregation function.

1775 If a declarative result aggregation is applied, it is still possible that no values can be provided for the
1776 aggregation of a particular output field, for example, if no subtask has set a value to an optional field (by
1777 omission or by an explicit `nil` value).

1778 In this case, the following rules determine how the aggregated output field of the parent task is set.

- 1779 • Rule (1): If the result value is optional (element defined with `minOccurs="0"` or attribute defined
1780 with `use="optional"`) then the corresponding element or attribute in the parent task output
1781 MUST be omitted.
- 1782 • Rule (2): If rule (1) does not apply and a default value is provided (element or attribute defined
1783 with `default="{value}"`) then the parent task output element or attribute MUST be explicitly
1784 set to this default value.
- 1785 • Rule (3): If rules (1)-(2) do not apply and the result value is a nillable element (element defined
1786 with `nillable="true"`) then the parent task output element MUST be set to a nil value (`<axsi:nil="true"/>`).
1787
- 1788 • Rule (4): If rules (1)-(3) do not apply, that is, the result is mandatory (element defined with
1789 `minOccurs="1"` or attribute defined with `use="required"`) but a value cannot be supplied,
1790 then a standard fault `htd:aggregationFailure` MUST be thrown to indicate a non-
1791 recoverable error.

1792 **Example:**

1793 Consider the following output document used in a parallel routing pattern:

```
1794 <element name="Award" type="tns:tAward" />  
1795 <complexType name="tAward">  
1796   <sequence>  
1797     <element name="AwardRecommended" type="xsd:string" />  
1798     <element name="AwardDetails" type="tns:tAwardDetails" />  
1799   </sequence>  
1800 </complexType>  
1801 <complexType name="tAwardDetails">  
1802   <sequence>  
1803     <element name="Amount" type="xsd:integer" />  
1804     <element name="Appraisal" type="xsd:string" />  
1805   </sequence>  
1806 </complexType>
```

1807 A possible result aggregation could then look like this. The first aggregation determines the most frequent
1808 occurrence of an award recommendation. The second aggregation calculates the average award amount
1809 for sub tasks with an award recommendation of 'yes'. The third aggregation creates a comma-separated
1810 concatenation of all sub task's appraisals.

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

```
1825     </htd:completion>
1826     </htd:completionBehavior>
1827 </htd:parallel>
```

1828 4.8.2.2 Explicit Result Assignment

1829 An `htd:copy` element describes the explicit assignment to an element of the parent task's output
1830 document.

```
1831 <htd:copy>+
1832   <htd:from expressionLanguage="anyURI"?>
1833     expression
1834   </htd:from>
1835   <htd:to part="NCName"? queryLanguage="anyURI"?>
1836     query
1837   </htd:to>
1838 </htd:copy>
```

1839 The `htd:copy` element is defined as follows:

- 1840 • The mandatory `htd:from` element MUST contain an expression used to calculate the result
1841 value. The expression can make use of WS-HumanTask aggregation functions.
- 1842 • The mandatory `htd:to` element MUST contain a query pointing to the location of an element of
1843 the tasks' output documents. This is the element created in the task's output document.

1844 Example 1:

1845 Consider the following output document used in a parallel routing pattern:

```
1846 <element name="Order" type="tns:tOrder" />
1847 <complexType name="tOrder">
1848   <sequence>
1849     <element name="Item" type="tns:tItem" maxOccurs="unbounded"/>
1850     <element name="TotalPrice" type="xsd:integer" />
1851   </sequence>
1852 </complexType>
1853 <complexType name="tItem">
1854   <sequence>
1855     ...
1856   </sequence>
1857 </complexType>
```

1858 A possible result aggregation could then look like this. All sub task order item lists are concatenated to
1859 one parent task order item list. The total price is calculated using an aggregation function.

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

```
1877     <htd:to>/Order/TotalPrice</htd:to>
1878     </htd:copy>
1879     </htd:result>
1880     </htd:completion>
1881     </htd:completionBehavior>
1882 </htd:parallel>
```

1883 **Example 2:**

1884 Output data from heterogeneous sub tasks is assigned into the parent task's output. The complete
1885 complex-typed sub task output documents are copied into child elements of the parent task output
1886 document.

```
1887 <htd:task name="bookTrip">
1888   ... produces itinerary ...
1889
1890   <htd:composition type="parallel" ...>
1891     <htd:subtask name="bookHotel">
1892       <htd:task>
1893         ... produces hotelReservation ...
1894       </htd:task>
1895     </htd:subtask>
1896     <htd:subtask name="bookFlight">
1897       <htd:task>
1898         ... produces flightReservation ...
1899       </htd:task>
1900     </htd:subtask>
1901   </htd:composition>
1902   ...
1903   <htd:completionBehavior>
1904     <htd:defaultCompletion>
1905       <htd:result>
1906         <htd:copy>
1907           <htd:from>
1908             htd:getSubtaskOutput("bookHotel",
1909                                   "bookHotelResponse",
1910                                   "/hotelReservation")
1911           </htd:from>
1912           <htd:to>/itinerary/hotelReservation</htd:to>
1913         </htd:copy>
1914         <htd:copy>
1915           <htd:from>
1916             htd:getSubtaskOutput("bookFlight",
1917                                   "bookFlightResponse",
1918                                   "/flightReservation")
1919           </htd:from>
1920           <htd:to>/itinerary/flightReservation</htd:to>
1921         </htd:copy>
1922       </htd:result>
1923     </htd:defaultCompletion>
1924   </htd:completionBehavior>
1925 </htd:task>
```

1926
1927

1928 4.9 Elements for Handling Timeouts and Escalations

1929 Timeouts and escalations allow the specification of a date or time before which the task or sub task has to
1930 reach a specific state. If the timeout occurs a set of actions is performed as the response. The state of the
1931 task is not changed. Several deadlines are specified which differ in the point when the timer clock starts
1932 and the state which has to be reached with the given duration or by the given date. They are:

- 1933 • Start deadline: Specifies the time until the task has to start, i.e. it has to reach state *InProgress*. It
1934 is defined as either the period of time or the point in time until the task has to reach state
1935 *InProgress*. Since expressions are allowed, durations and deadlines can be calculated at runtime,
1936 which for example enables custom calendar integration. The time starts to be measured from the
1937 time at which the task enters the state *Created*. If the task does not reach state *InProgress* by the
1938 deadline an escalation action or a set of escalation actions is performed. Once the task is started,
1939 the timer becomes obsolete.
- 1940 • Completion deadline: Specifies the due time of the task. It is defined as either the period of time
1941 until the task gets due or the point in time when the task gets due. The time starts to be measured
1942 from the time at which the task enters the state *Created*. If the task does not reach one of the final
1943 states (*Completed*, *Failed*, *Error*, *Exited*, *Obsolete*) by the deadline an escalation action or a set
1944 of escalation actions is performed.

1945 The element `<deadlines>` is used to include the definition of all deadlines within the task definition. It is
1946 optional. If present then the WS-HumanTask Definition MUST specify at least one deadline. Deadlines
1947 defined in ad-hoc sub tasks created at runtime MUST NOT contradict the deadlines of their parent task.
1948 The value of the name attribute MUST be unique for all deadline specifications within a task definition.

1949 Syntax:

```
1950 <htd:deadlines>  
1951  
1952   <htd:startDeadline name="NCName"*  
1953  
1954     <htd:documentation xml:lang="xsd:language"? >*<br>1955     text<br>1956   </htd:documentation>  
1957  
1958   ( <htd:for expressionLanguage="anyURI"? >  
1959     duration-expression<br>1960   </htd:for>  
1961   | <htd:until expressionLanguage="anyURI"? >  
1962     deadline-expression<br>1963   </htd:until>  
1964   )  
1965  
1966   <htd:escalation name="NCName"*  
1967     ...  
1968   </htd:escalation>  
1969 </htd:startDeadline>  
1970  
1971   <htd:completionDeadline name="NCName"*  
1972     ...  
1973   </htd:completionDeadline>  
1974 </htd:deadlines>
```

1977 The language used in expressions is specified using the `expressionLanguage` attribute. This attribute
1978 is optional. If not specified, the default language as inherited from the closest enclosing element that
1979 specifies the attribute is used.

1980 For all deadlines if a status is not reached within a certain time then an escalation action, specified using
1981 element `<escalation>`, can be triggered. The `<escalation>` element is defined in the section below.
1982 When the task reaches a final state (*Completed, Failed, Error, Exited, Obsolete*) all deadlines are deleted.
1983 Escalations are triggered if

- 1984 1. The associated point in time is reached, or duration has elapsed, and
- 1985 2. The associated condition (if any) evaluates to true

1986 Escalations use notifications to inform people about the status of the task. Optionally, a task might be
1987 reassigned to some other person or group as part of the escalation. Notifications are explained in more
1988 detail in section 6 “Notifications”. For an escalation, a WS-HumanTask Definition MUST specify exactly
1989 one escalation action.

1990 When defining escalations, a notification can be either referred to, or defined inline.

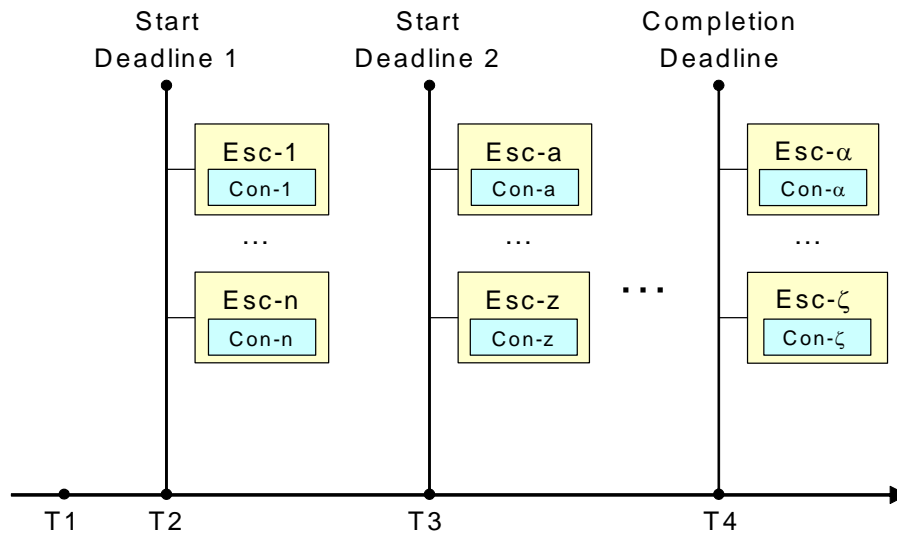
- 1991 • A notification defined in the `<humanInteractions>` root element or imported from a different
1992 namespace can be referenced by specifying its QName in the `reference` attribute of a
1993 `<localNotification>` element. When referring to a notification, the priority and the people
1994 assignments of the original notification definition MAY be overridden using the elements
1995 `<priority>` and `<peopleAssignments>` contained in the `<localNotification>` element.
- 1996 • An inlined notification is defined by a `<notification>` element.

1997 Notifications used in escalations can use the same type of input data as the surrounding task or sub task,
1998 or different type of data. If the same type of data is used then the input message of the task or sub task is
1999 passed to the notification implicitly. If not, then the `<toPart>` elements are used to assign appropriate
2000 data to the notification, i.e. to explicitly create a multi-part WSDL message from the data. The `part`
2001 attribute refers to a part of the WSDL message. The `expressionLanguage` attribute specifies the
2002 language used in the expression. The attribute is optional. If not specified, the default language as
2003 inherited from the closest enclosing element that specifies the attribute is used.

2004 A WS-HumanTask Definition MUST specify a `<toPart>` element for every part in the WSDL message
2005 definition because parts not explicitly represented by `<toPart>` elements would result in uninitialized parts
2006 in the target WSDL message. The order in which parts are specified is not relevant. If multiple `<toPart>`
2007 elements are present, a WS-HumanTask Processor MUST execute them in an “all or nothing” manner. If
2008 any of the `<toPart>`s fails, the escalation action will not be performed and the execution of the task is not
2009 affected.

2010 Reassignments are used to replace the potential owners of a task when an escalation is triggered. The
2011 `<reassignment>` element is used to specify reassignment. If present then a WS-HumanTask Definition
2012 MUST specify potential owners. A reassignment triggered by a sub task escalation MUST apply to the
2013 sub task only. A reassignment MAY comprise of a complex people assignment using Routing Patterns.

2014 In the case where several reassignment escalations are triggered, the first reassignment (lexical order)
2015 MUST be considered for execution by the WS-HumanTask Processor. The task is set to state *Ready* after
2016 reassignment. Reassignments and notifications are performed in the lexical order.



2017

2018 A task MAY have multiple start deadlines and completion deadlines associated with it. Each such
 2019 deadline encompasses escalation actions each of which MAY send notifications to certain people. The
 2020 corresponding set of people MAY overlap.

2021 As an example, the figure depicts a task that has been created at time T1. Its two start deadlines would
 2022 be missed at time T2 and T3, respectively. The associated escalations whose conditions evaluate to
 2023 "true" are triggered. Both, the escalations Esc-1 to Esc-n as well as escalations Esc-a to Esc-z can
 2024 involve an overlapping set of people. The completion deadline would be missed at time T4.

2025 **Syntax:**

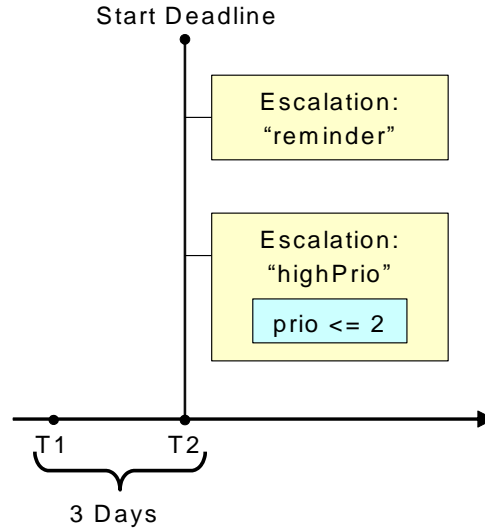
```

2026 <htd:deadlines>
2027
2028   <htd:startDeadline name="NCName"*
2029     ...
2030     <htd:escalation name="NCName"*
2031
2032       <htd:condition expressionLanguage="anyURI"?>?
2033         boolean-expression
2034       </htd:condition>
2035
2036       <htd:toParts?>
2037         <htd:toPart part="NCName"
2038           expressionLanguage="anyURI"?>+
2039           expression
2040         </htd:toPart>
2041       </htd:toParts>
2042
2043       <!-- notification specified by reference -->
2044       <htd:localNotification reference="QName"?
2045         <htd:priority expressionLanguage="anyURI"?>?
2046         integer-expression
2047       </htd:priority>
2048       <htd:peopleAssignments?>
2049         <htd:recipients>
2050           ...
2051         </htd:recipients>
2052       </htd:peopleAssignments>
  
```

```
2053
2054     </htd:localNotification>
2055
2056     <!-- notification specified inline -->
2057     <htd:notification name="NCName"?
2058         ...
2059     </htd:notification>
2060
2061     <htd:reassignment?
2062         <htd:potentialOwners>
2063             ...
2064         </htd:potentialOwners>
2065     </htd:reassignment>
2066
2067 </htd:escalation>
2068
2069 </htd:startDeadline>
2070
2071 <htd:completionDeadline name="NCName"*
2072     ...
2073 </htd:completionDeadline>
2074
2075 </htd:deadlines>
2076
2077
2078
```


2079 **Example:**

2080 The following example shows the specification of a start deadline with escalations. At runtime, the
2081 following picture depicts the result of what is specified in the example:



2082 The human task is created at T1. If it has not been started, i.e., no person is working on it until T2, then
2083 the escalation “reminder” is triggered that notifies the potential owners of the task that work is waiting for
2084 them. In case the task has high priority then at the same time the regional manager is informed. If the
2085 task amount is greater than or equal 10000 the task is reassigned to Alan.

2086 In case that task has been started before T2 was reached, then the start deadline is deactivated, no
2087 escalation occurs.

```
2088 <htd:startDeadline name="sendNotifications">
2089   <htd:documentation xml:lang="en-US">
2090     If not started within 3 days, - escalation notifications are sent
2091     if the claimed amount is less than 10000 - to the task's potential
2092     owners to remind them or their todo - to the regional manager, if
2093     this approval is of high priority (0,1, or 2) - the task is
2094     reassigned to Alan if the claimed amount is greater than or equal
2095     10000
2096   </htd:documentation>
2097   <htd:for>P3D</htd:for>
2098   <htd:escalation name="reminder">
2099
2100     <htd:condition>
2101       <![CDATA[
2102         htd:getInput("ClaimApprovalRequest")/amount < 10000
2103       ]]>
2104     </htd:condition>
2105
2106     <htd:toParts>
2107       <htd:toPart name="firstname">
2108         htd:getInput("ClaimApprovalRequest","ApproveClaim")/firstname
2109       </htd:toPart>
2110       <htd:toPart name="lastname">
2111         htd:getInput("ClaimApprovalRequest","ApproveClaim")/lastname
2112       </htd:toPart>
2113     </htd:toParts>
2114
```

```

2115 <htd:localNotification reference="tns:ClaimApprovalReminder">
2116
2117   <htd:documentation xml:lang="en-US">
2118     Reuse the predefined notification "ClaimApprovalReminder".
2119     Overwrite the recipients with the task's potential owners.
2120   </htd:documentation>
2121
2122   <htd:peopleAssignments>
2123     <htd:recipients>
2124       <htd:from>htd:getPotentialOwners("ApproveClaim")</htd:from>
2125     </htd:recipients>
2126   </htd:peopleAssignments>
2127
2128 </htd:localNotification>
2129
2130 </htd:escalation>
2131
2132 <htd:escalation name="highPrio">
2133
2134   <htd:condition>
2135     <![CDATA[
2136       (htd:getInput("ClaimApprovalRequest")/amount < 10000
2137       && htd:getInput("ClaimApprovalRequest")/prio <= 2)
2138     ]]>
2139   </htd:condition>
2140
2141   <!-- task input implicitly passed to the notification -->
2142
2143   <htd:notification name="ClaimApprovalOverdue">
2144     <htd:documentation xml:lang="en-US">
2145       An inline defined notification using the approval data as its
2146       input.
2147     </htd:documentation>
2148
2149     <htd:interface portType="tns:ClaimsHandlingPT"
2150       operation="escalate" />
2151
2152     <htd:peopleAssignments>
2153       <htd:recipients>
2154         <htd:from logicalPeopleGroup="regionalManager">
2155           <htd:argument name="region">
2156             htd:getInput("ClaimApprovalRequest")/region
2157           </htd:argument>
2158         </htd:from>
2159       </htd:recipients>
2160     </htd:peopleAssignments>
2161
2162     <htd:presentationElements>
2163       <htd:name xml:lang="en-US">Claim approval overdue</htd:name>
2164       <htd:name xml:lang="de-DE">
2165         Überfällige Schadensforderungsgenehmigung
2166       </htd:name>
2167     </htd:presentationElements>
2168
2169   </htd:notification>
2170
2171 </htd:escalation>
2172

```

```
2173 <htd:escalation name="highAmountReassign">
2174
2175   <htd:condition>
2176     <![CDATA[
2177       htd:getInput("ClaimApprovalRequest")/amount >= 10000
2178     ]]>
2179   </htd:condition>
2180
2181   <htd:reassignment>
2182     <htd:documentation>
2183       Reassign task to Alan if amount is greater than or equal
2184       10000.
2185     </htd:documentation>
2186
2187     <htd:potentialOwners>
2188       <htd:from>
2189         <htd:literal>
2190           <htt:organizationalEntity>
2191             <htt:user>Alan</htt:user>
2192           </htt:organizationalEntity>
2193         </htd:literal>
2194       </htd:from>
2195     </htd:potentialOwners>
2196
2197   </htd:reassignment>
2198
2199 </htd:escalation>
2200
2201 </htd:startDeadline>
```

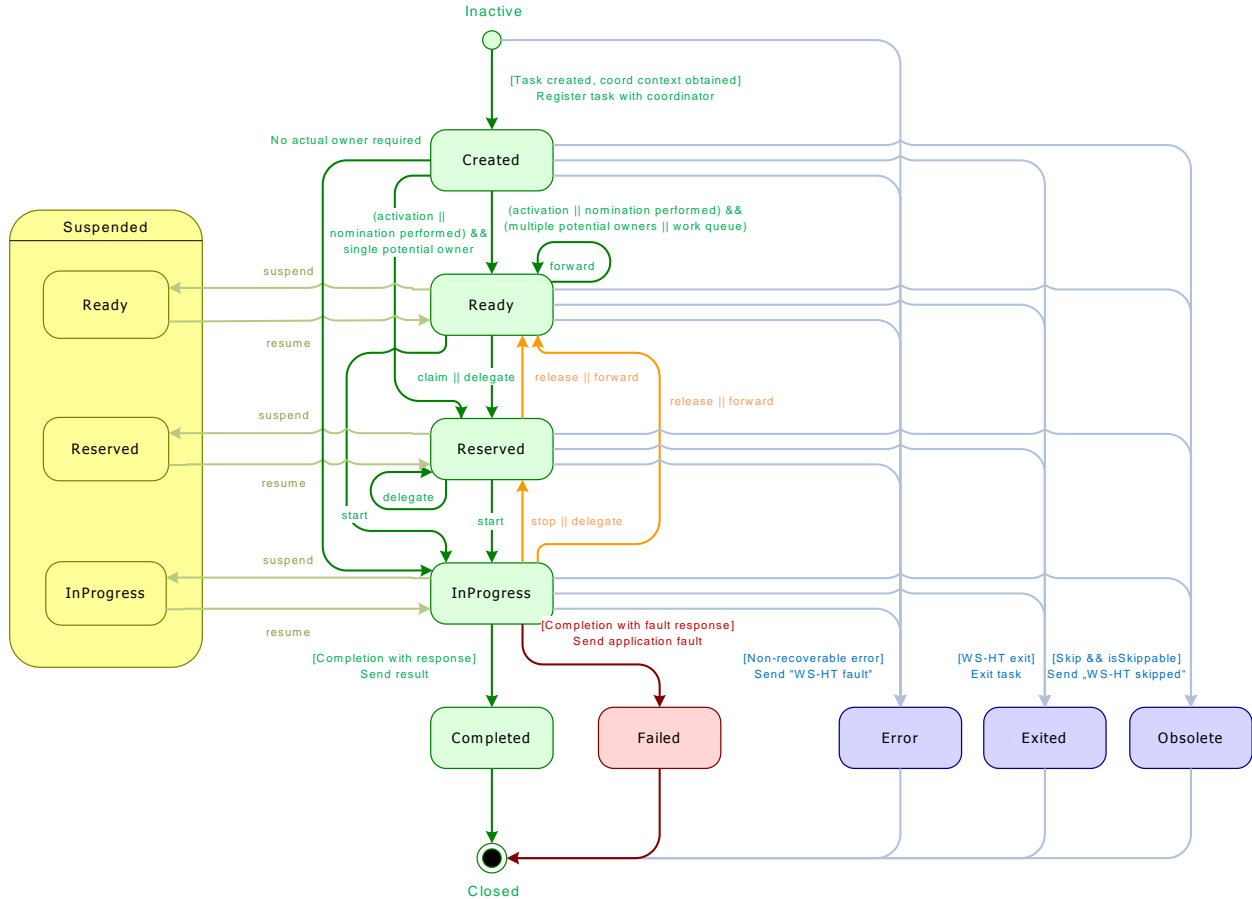
2202 All timeouts and escalations apply to sub tasks also. If htd:escalation is triggered for a sub task, then any
2203 htd:reassignment MUST be applied only to that.

2204

2205

2206 4.10 Human Task Behavior and State Transitions

2207 Human tasks can have a number of different states and substates. The state diagram for human tasks
2208 below shows the different states and the transitions between them.



2209

2210 4.10.1 Normal processing of a Human Task

2211 Upon creation, a task goes into its initial state *Created*. Task creation starts with the initialization of its
2212 properties in the following order:

- 2213 1. Input message
- 2214 2. Priority
- 2215 3. Generic human roles (such as excluded owners, potential owners and business administrators)
2216 are made available in the lexical order of their definition in the people assignment definition with
2217 the constraint that excluded owners are taken into account when evaluating the potential owners.
- 2218 4. All other properties are evaluated after these properties in an implementation dependent order.

2219 Task creation succeeds irrespective of whether the people assignment returns a set of values or an
2220 empty set. People queries that cannot be executed successfully are treated as if they were returning an
2221 empty set.

2222 If potential owners were not assigned automatically during task creation then they **MUST** be assigned
2223 explicitly using nomination, which is performed by the task's business administrator. The result of
2224 evaluating potential owners removes the excluded owners from results. The task remains in the state
2225 *Created* until it is activated (i.e., an activation timer has been specified) and has potential owners.

2226 When the task has a single potential owner, it transitions into the *Reserved* state, indicating that it is
2227 assigned to a single actual owner. Otherwise (i.e., when it has multiple potential owners or is assigned to
2228 a work queue), it transitions into the *Ready* state, indicating that it can be claimed by one of its potential
2229 owners. Once a potential owner claims the task, it transitions into the *Reserved* state, making that
2230 potential owner the actual owner.

2231 Once work is started on a task that is in state *Ready* or *Reserved*, it goes into the *InProgress* state,
2232 indicating that it is being worked on – if the transition is from *Ready*, the user starting the work becomes
2233 its actual owner.

2234 On successful completion of the work, the task transitions into the *Completed* final state. On unsuccessful
2235 completion of the work (i.e., with an exception), the task transitions into the *Failed* final state.

2236 The lifecycle of sub tasks is the same as that of the main task.

2237 For human tasks that have subtasks two different cases exist, with different implications:

2238 1. Tasks with subtasks where an actual owner is required

2239 2. Tasks with subtasks where no actual owner is required

2240 The first case has the sub-case where a potential owner has been modeled on the primary task and
2241 subtasks have been modeled that are activated either manually or automatically. Another sub-case of the
2242 first case is the one where no potential owner has been modeled and thus nomination has to occur. In all
2243 cases there is an actual owner eventually and the primary task goes through the state transitions from
2244 *Created* to *Ready* to *Reserved* to *InProgress*, etc.

2245 In the second case where no actual owner is desired the human task (the primary task) directly transitions
2246 from state *Created* to *InProgress*. Subtasks are always instantiated automatically.

2247 **4.10.2 Releasing a Human Task**

2248 The current actual owner of a human task can *release* a task to again make it available for all potential
2249 owners. A task can be released from active states that have an actual owner (*Reserved*, *InProgress*),
2250 transitioning it into the *Ready* state. Business data associated with the task (intermediate result data, ad-
2251 hoc attachments and comments) is kept.

2252 A task that is currently *InProgress* can be stopped by the actual owner, transitioning it into state
2253 *Reserved*. Business data associated with the task as well as its actual owner is kept.

2254 **4.10.3 Delegating or Forwarding a Human Task**

2255 Task's potential owners, actual owner or business administrator can *delegate* a task to another user,
2256 making that user the actual owner of the task, and also adding her to the list of potential owners in case
2257 she is not, yet. A task can be delegated when it is in an active state (*Ready*, *Reserved*, *InProgress*), and
2258 transitions the task into the *Reserved* state. Business data associated with the task is kept.

2259 Similarly, task's potential owners, actual owner or business administrator can forward an active task to
2260 another person or a set of people, replacing himself by those people in the list of potential owners.
2261 Potential owners can only forward tasks that are in the *Ready* state. Forwarding is possible if the task has
2262 a set of individually assigned potential owners, not if its potential owners are assigned using one or many
2263 groups. If the task is in the *Reserved* or *InProgress* state then the task is implicitly released first, that is,
2264 the task is transitioned into the *Ready* state. Business data associated with the task is kept. The user
2265 performing the forward is removed from the set of potential owners of the task, and the forwarder is
2266 added to the set of potential owners.

2267 **4.10.4 Sub Task Event Propagation**

2268 Task state transitions may be caused by the invocation of API operations (see section 7 "Programming
2269 Interfaces") or by events (see section 8 "Interoperable Protocol for Advanced Interaction with Human
2270 Tasks").

2271 If a task has sub tasks then some state transitions are propagated to these sub tasks. Conversely, if a
 2272 task has a parent task then some state transitions are propagated to that parent task.

2273 The following table defines how task state transitions MUST be propagated to sub tasks and to parent
 2274 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”)

2275 All other task state transitions MUST NOT affect sub tasks or a parent task.

2276 **4.11 History of a Human Task**

2277 Task lifecycle state changes and data changes are maintained as a history of task events. Task events
 2278 contain the following data:

2279 **Task Event**

- 2280 • event id
- 2281 • event time
- 2282 • task id
- 2283 • user (principal) that caused the state change
- 2284 • event type (e.g. claim task).
- 2285 • event data (e.g. data used in setOutput) and fault name (event was setFault)
- 2286 • startOwner - the actual owner before the event.
- 2287 • endOwner - the actual owner after the event.
- 2288 • task status at the end of the event

2289 For example, if the User1 delegated a task to User2, then the user and startOwner would be User1,
 2290 endOwner would be User2. The event data would be the <htt:organizationalEntity/> element used in the
 2291 WSHT delegate operation.

2292 The system generated attribute 'event id' MUST be unique on a per task basis.

2293 **4.11.1 Task Event Types and Data**

2294 Some task events (e.g. setOutput) may have data associated with event and others may not (e.g. claim).
 2295 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/>

Actions/Operations resulting in task events			
Event Type	Owner Change	State Change	Data Value
			</htt:addAttachment>
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			

2296 4.11.2 Retrieving the History

2297 There is a `getTaskHistory` operation that allows a client to query the system and retrieve a list of task
2298 events that represent the history of the task. This operation can:

- 2299 • Return a list of task events with optional data
- 2300 • Return a list of task events without optional event data
- 2301 • Return a subset of the events based on a range (for paging)
- 2302 • Return a filtered list of events.

2303 The option to whether or not to include event data is useful since in some cases the event data content
2304 (e.g. `setOutput`) may be large. In a typical case, an API client should be able to query the system to get a
2305 "light weight" response of events (e.g. with out event data) and then when necessary, make an additional
2306 API call to get a specific event details with data. The latter can be accomplished by specifying the event id
2307 when invoking the `getTaskHistory` operation.

2308 The XML Schema definition of the filter is the following:

```

2309 <xsd:complexType name="tTaskHistoryFilter">
2310   <xsd:choice>
2311     <xsd:element name="eventId" type="xsd:integer" />
2312     <!-- Filter to allow narrow down query by status, principal,
2313          event Type. -->
2314     <xsd:sequence>
2315       <xsd:element name="status" type="tStatus" minOccurs="0"
2316                   maxOccurs="unbounded" />
2317       <xsd:element name="eventType" type="tTaskEventType" minOccurs="0"
2318                   maxOccurs="unbounded" />
2319       <xsd:element name="principal" type="xsd:string" minOccurs="0" />
2320       <xsd:element name="afterEventTime" type="xsd:dateTime"
2321                   minOccurs="0" />
2322       <xsd:element name="beforeEventTime" type="xsd:dateTime"
2323                   minOccurs="0" />
2324     </xsd:sequence>
2325   </xsd:choice>
2326 </xsd:complexType>
2327
2328 <xsd:simpleType name="tTaskEventType">
2329   <xsd:restriction base="xsd:string">
2330     <xsd:enumeration value="create" />
2331     <xsd:enumeration value="claim" />
2332     <xsd:enumeration value="start" />
2333     <xsd:enumeration value="stop" />
2334     <xsd:enumeration value="release" />
2335     <xsd:enumeration value="suspend" />

```

```

2336 <xsd:enumeration value="suspendUntil" />
2337 <xsd:enumeration value="resume" />
2338 <xsd:enumeration value="complete" />
2339 <xsd:enumeration value="remove" />
2340 <xsd:enumeration value="fail" />
2341 <xsd:enumeration value="setPriority" />
2342 <xsd:enumeration value="addAttachment" />
2343 <xsd:enumeration value="deleteAttachment" />
2344 <xsd:enumeration value="addComment" />
2345 <xsd:enumeration value="updateComment" />
2346 <xsd:enumeration value="deleteComment" />
2347 <xsd:enumeration value="skip" />
2348 <xsd:enumeration value="forward" />
2349 <xsd:enumeration value="delegate" />
2350 <xsd:enumeration value="setOutput" />
2351 <xsd:enumeration value="deleteOutput" />
2352 <xsd:enumeration value="setFault" />
2353 <xsd:enumeration value="deleteFault" />
2354 <xsd:enumeration value="activate" />
2355 <xsd:enumeration value="nominate" />
2356 <xsd:enumeration value="setGenericHumanRole" />
2357 <xsd:enumeration value="expire" />
2358 <xsd:enumeration value="escalated" />
2359 </xsd:restriction>
2360 </xsd:simpleType>

```

2361 The XML Schema definition of events returned for the history is the following:

```

2362 <xsd:element name="taskEvent">
2363   <xsd:complexType>
2364     <xsd:annotation>
2365       <xsd:documentation>
2366         A detailed event that represents a change in the task's state.
2367       </xsd:documentation>
2368     </xsd:annotation>
2369     <xsd:sequence>
2370       <!-- event id - unique per task -->
2371       <xsd:element name="id" type="xsd:integer" />
2372       <!-- event date time -->
2373       <xsd:element name="eventTime" type="xsd:dateTime" />
2374       <!-- task ID -->
2375       <xsd:element name="identifier" type="xsd:anyURI" />
2376       <xsd:element name="principal" type="xsd:string" minOccurs="0"
2377         nillable="true" />
2378       <!-- Event type. Note - using a restricted type limits
2379         extensibility to add custom event types. -->
2380       <xsd:element name="eventType" type="tTaskEventType" />
2381       <!-- actual owner of the task before the event -->
2382       <xsd:element name="startOwner" type="xsd:string" minOccurs="0"
2383         nillable="true" />
2384       <!-- actual owner of the task after the event -->
2385       <xsd:element name="endOwner" type="xsd:string" minOccurs="0"
2386         nillable="true" />
2387       <!-- WSH task status -->
2388       <xsd:element name="status" type="tStatus" />
2389       <!-- boolean to indicate this event has optional data -->
2390       <xsd:element name="hasData" type="xsd:boolean" minOccurs="0" />
2391       <xsd:element name="eventData" type="xsd:anyType" minOccurs="0"

```

```
2392         nillable="true" />
2393     <xsd:element name="faultName" type="xsd:string" minOccurs="0"
2394         nillable="true" />
2395     <!-- extensibility -->
2396     <xsd:any namespace="##other" processContents="lax" minOccurs="0"
2397         maxOccurs="unbounded" />
2398 </xsd:sequence>
2399 </xsd:complexType>
2400 </xsd:element>
2401
```

2402 5 Lean Tasks

2403 The <leanTask> element is used to specify human tasks. This section introduces the syntax for the
2404 element, and individual properties are explained in subsequent sections.

2405 5.1 Overall Syntax

2406 The element <leanTask> derives from the type htd:tTask, with the following augmentations:

```
2407 <htd:leanTask>  
2408   <htd:interface>...</htd:interface>  
2409   <htd:messageSchema>...</htd:messageSchema>  
2410   ... All elements from htd:task except <interface> and <composition> ...  
2411   <htd:composition>...</htd:composition>  
2412 </htd:leanTask>
```

2413 5.2 Properties

2414 The following attributes and elements are defined for lean tasks and are different from the definition of
2415 htd:task:

- 2416 • `interface` – Lean tasks are created through the CreateLeanTask operation (section 7.3.4), and
2417 their input message is derived from the messageSchema element. Therefore, an interface
2418 element might contradict that information, and to prevent that, interface is banned.
- 2419 • `messageSchema` – Identifies the schema of the inputMessage and outputMessage for the lean
2420 task, and if the renderings element is not defined, the WS-HumanTask Processor can use this to
2421 generate a rendering or pass this data directly to a WS-HumanTask Client such that the
2422 rendering is generated from the messageSchema.
- 2423 • `composition` – Lean tasks cannot have explicitly declared subtasks as defined for composite
2424 tasks (section 4.6), consequently, this element is banned.

2425 5.3 Message Schema

2426 This element references the schema of the data that is used for both the input and output messages of
2427 the lean task.

```
2428 <messageSchema>  
2429   <messageField name="xsd:NCName" type="xsd:QName"*>  
2430     <messageDisplay xml:lang="xsd:language"?>+  
2431       Language specific display  
2432     </messageDisplay>  
2433     <messageChoice value="xsd:anySimpleType"*>  
2434       <messageDisplay xml:lang="xsd:language"?>+  
2435         Language specific display  
2436       </messageDisplay>  
2437     </messageChoice>  
2438   </messageField>  
2439 </messageSchema>
```

2440 The <messageSchema> element specifies the data that a Lean Task accepts. As it is currently defined, a
2441 WS-HumanTask Processor could render the following form elements in a way that only requires vendor-
2442 specific knowledge between the WS-HumanTask Processor and the WS-HumanTask Client and no
2443 vender-specific knowledge between the WS-HumanTask Processor and the WS-HumanTask Parent:

- 2444 • String
- 2445 • Integer

- 2446 • Float
- 2447 • Date Time
- 2448 • Bool
- 2449 • Enumeration (Choice)

2450 Each of these is accomplished by using an instance of a `<messageField>`. For string, integer, float,
2451 datetime, and boolean fields, this is accomplished by using the type attribute of the `<messageField>`.
2452 The supported set of values are: `xsd:string`, `xsd:integer`, `xsd:float`, `xsd:datetime`, and
2453 `xsd:boolean`, all respectively matching the list above. If a simple rendering language like HTML were
2454 used, this could be accomplished by using a textbox control that simply had special rules about the format
2455 of its input.

2456 The enumeration field uses a combination of one `<messageField>` element and possibly many child
2457 `<messageChoice>` elements. Each child `<messageChoice>` represents one possible option that could
2458 be selected from the enumeration. If a simple rendering language like HTML were used, this could be
2459 shown using radio buttons, a dropdown list, or a listbox that only supports single selection.

2460 For all `<messageField>` and `<messageChoice>` elements, it is possible to specify a per-language
2461 `<messageDisplay>` element. It uses `xml:lang`, a standard XML attribute, to define the language of the
2462 enclosed information. This attribute uses tags according to RFC 1766 (see [RFC1766]). There could be
2463 zero or more `<messageDisplay>` elements. A `<messageField>` or `<messageChoice>` MUST NOT
2464 specify multiple `<messageDisplay>` elements having the same value for the attribute `xml:lang`.

2465 The combination of `<messageSchema>` and `<possibleOutcomes>` can be used to generate a form of
2466 sufficient functionality for many simple tasks, precluding the need for a renderings element.

2467 **Example:**

```
2468 <messageSchema>  
2469   <messageField name="amount" type="xsd:float">  
2470     <messageDisplay xml:lang="en-us">Amount</messageDisplay>  
2471     <messageDisplay xml:lang="fr-fr">Quantité</messageDisplay>  
2472   </messageField>  
2473   <messageField name="currencyUnit" type="xsd:string">  
2474     <messageDisplay xml:lang="en-us">Currency</messageDisplay>  
2475     <messageDisplay xml:lang="fr-fr">Devise</messageDisplay>  
2476     <messageChoice value="USD">  
2477       <messageDisplay xml:lang="en-us">US Dollars</messageDisplay>  
2478       <messageDisplay xml:lang="fr-fr">US Dollars</messageDisplay>  
2479     </messageChoice>  
2480     <messageChoice value="EURO">  
2481       <messageDisplay xml:lang="en-us">Euro Dollars</messageDisplay>  
2482       <messageDisplay xml:lang="fr-fr">Euros</messageDisplay>  
2483     </messageChoice>  
2484   </messageField>  
2485 </messageSchema>
```

2486

2487

2488 5.4 Example: ToDoTask

2489 The following XML could be used for a simple 'ToDoTask':

```
2490 <htd:task name="ToDoTask">
2491   <htd:messageSchema />
2492   <htd:possibleOutcomes>
2493     <htd:possibleOutcome name="Completed" />
2494     ... language specific translations ...
2495   </htd:possibleOutcomes>
2496   <htd:delegation potentialDelegates="anybody" />
2497   <htd:presentationElements>
2498     <htd:name>ToDo Task</htd:name>
2499     ... language specific translations ...
2500     <htd:subject>Please complete the described work</htd:subject>
2501     ... language specific translations ...
2502     <htd:description contentType="mimeTypeString" />
2503     ... language specific translations ...
2504   </htd:presentationElements>
2505 </htd:task>
```


2506

6 Notifications

2507 Notifications are used to notify a person or a group of people of a noteworthy business event, such as
2508 that a particular order has been approved, or a particular product is about to be shipped. They are also
2509 used in escalation actions to notify a user that a task is overdue or a task has not been started yet. The
2510 person or people to whom the notification will be assigned to could be provided, for example, as result of
2511 a people query to organizational model.

2512 Notifications are simple human interactions that do not block the progress of the caller, that is, the caller
2513 does not wait for the notification to be completed. Moreover, the caller cannot influence the execution of
2514 notifications, e.g. notifications are not terminated if the caller terminates. The caller, i.e. an application, a
2515 business process or an escalation action, initiates a notification passing the required notification data. The
2516 notification appears on the task list of all notification recipients. After a notification recipient removes it,
2517 the notification disappears from the recipient's task list.

2518 A notification MAY have multiple recipients and optionally one or many business administrators. The
2519 generic human roles task initiator, task stakeholders, potential owners, actual owner and excluded
2520 owners play no role.

2521 Presentation elements and task rendering, as described in sections 4.3 and 4.4 respectively, are used for
2522 notifications also. In most cases the subject line and description are sufficient information for the
2523 recipients, especially if the notifications are received in an e-mail client or mobile device. But in some
2524 cases the notifications can be received in a proprietary client so the notification can support a proprietary
2525 rendering format to enable this to be utilized to the full, such as for rendering data associated with the
2526 caller invoking the notification. For example, the description could include a link to the process audit trail
2527 or a button to navigate to business transactions involved in the underlying process.

2528 Notifications do not have ad-hoc attachments, comments or deadlines.

6.1 Overall Syntax

2529 Definition of notifications

```
2530 <htd:notification name="NCName">  
2531  
2532   <htd:interface portType="QName" operation="NCName"/>  
2533  
2534   <htd:priority expressionLanguage="anyURI"??>  
2535     integer-expression  
2536   </htd:priority>  
2537  
2538   <htd:peopleAssignments>  
2539  
2540     <htd:recipients>  
2541       ...  
2542     </htd:recipients>  
2543  
2544     <htd:businessAdministrators??>  
2545       ...  
2546     </htd:businessAdministrators>  
2547  
2548   </htd:peopleAssignments>  
2549  
2550   <htd:presentationElements>  
2551     ...  
2552   </htd:presentationElements>  
2553  
2554   <htd:renderings??>
```

```
2556 ...
2557 </htd:renderings>
2558
2559 </htd:notification>
```

2560 6.2 Properties

2561 The following attributes and elements are defined for notifications:

- 2562 • `name`: This attribute is used to specify the name of the notification. The name combined with the
2563 target namespace MUST uniquely identify a notification in the notification definition. The attribute
2564 is mandatory. It is not used for notification rendering.
- 2565 • `interface`: This element is used to specify the operation used to invoke the notification. The
2566 operation is specified using WSDL, that is a WSDL port type and WSDL operation are defined.
2567 The element and its `portType` and `operation` attributes are mandatory. In the `operation`
2568 attribute, a WS-HumanTask Definition MUST reference a one-way WSDL operation.
- 2569 • `priority`: This element is used to specify the priority of the notification. It is an optional
2570 element which value is an integer expression. If present then the WS-HumanTask Definition
2571 MUST specify a value between 0 and 10, where 0 is the highest priority and 10 is the lowest. If
2572 not present, the priority of the notification is considered as 5. The result of the expression
2573 evaluation is of type `http:tPriority`. The `expressionLanguage` attribute specifies the
2574 language used in the expression. The attribute is optional. If not specified, the default language
2575 as inherited from the closest enclosing element that specifies the attribute is used.
- 2576 • `peopleAssignments`: This element is used to specify people assigned to the notification. The
2577 element is mandatory. A WS-HumanTask Definition MUST include a people assignment for
2578 recipients and MAY include a people assignment for business administrators.
- 2579 • `presentationElements`: The element is used to specify different information used to display
2580 the notification, such as name, subject and description, in a task list. The element is mandatory.
2581 See section 4.3 for more information on presentation elements.
- 2582 • `rendering`: The element is used to specify rendering method. It is optional. If not present,
2583 notification rendering is implementation dependent. See section 4.4 for more information on
2584 rendering.

2585 6.3 Notification Behavior and State Transitions

2586 Same as human tasks, notifications are in pseudo-state *Inactive* before they are activated. Once they are
2587 activated they move to the *Ready* state. This state is observable, that is, when querying for notifications
2588 then all notifications in state *Ready* are returned. When a notification is removed then it moves into the
2589 final pseudo-state *Removed*.

2590 7 Programming Interfaces

2591 7.1 Operations for Client Applications

2592 A number of applications are involved in the life cycle of a task. These comprise:

- 2593 • The task list client, i.e. a client capable of displaying information about the task under
2594 consideration
- 2595 • The requesting application, i.e. any partner that has initiated the task
- 2596 • The supporting application, i.e. an application launched by the task list client to support
2597 processing of the task.

2598 The task infrastructure provides access to a given task. It is important to understand that what is meant
2599 by *task list client* is the software that presents a UI to one authenticated user, irrespective of whether this
2600 UI is rendered by software running on server hardware (such as in a portals environment) or client
2601 software (such as a client program running on a users workstation or PC).

2602 A given task exposes a set of operations to this end. A WS-HumanTask Processor MUST provide the
2603 operations listed below and an application (such as a task list client) can use these operations to
2604 manipulate the task. All operations MUST be executed in a synchronous fashion and MUST return a fault
2605 if certain preconditions do not hold. For operations that are not expected to return a response they MAY
2606 return a void message. The above applies to notifications also.

2607 An operation takes a well-defined set of parameters as its input. Passing an illegal parameter or an illegal
2608 number of parameters MUST result in the `hta:illegalArgumentFault` being returned. Invoking an
2609 operation that is not allowed in the current state of the task MUST result in an
2610 `hta:illegalStateFault`.

2611 By default, the identity of the person on behalf of which the operation is invoked is passed to the task.
2612 When the person is not authorized to perform the operation the `hta:illegalAccessFault` and
2613 `hta:recipientNotAllowed` MUST be returned in the case of tasks and notifications respectively.

2614 Invoking an operation that does not apply to the task type (e.g., invoking claim on a notification) MUST
2615 result in an `hta:illegalOperationFault`.

2616 The language of the person on behalf of which the operation is invoked is assumed to be available to
2617 operations requiring that information, e.g., when accessing presentation elements.

2618 For an overview of which operations are allowed in what state, refer to section 4.10 "Human Task
2619 Behavior and State Transitions". For a formal definition of the allowed operations, see Appendix D "WS-
2620 HumanTask Client API Port Type".

2621 For information which generic human roles are authorized to perform which operations, refer to section
2622 7.1.4 "Operation Authorizations".

2623 This specification does not stipulate the authentication, language passing, addressing, and binding
2624 scheme employed when calling an operation. This can be achieved using different mechanisms (e.g. WS-
2625 Security, WS-Addressing).

2626 7.1.1 Participant Operations

2627 Operations are executed by end users, i.e. actual or potential owners. The identity of the user is implicitly
2628 passed when invoking any of the operations listed in the table below.

2629 If the task is in a predefined state listed as valid pre-state before the operation is invoked then, upon
2630 successful completion, the task MUST be in the post state defined for the operation. If the task is in a
2631 predefined state that is not listed as valid pre-state before the operation is invoked then the operation
2632 MUST be rejected and MUST NOT cause a task state transition.

2633 All of the operations below apply to tasks and sub tasks only unless specifically noted below.

2634 The column “**Supports Batch Processing**” below indicates if an operation can be used to process
 2635 multiple human tasks at the same time. One or more operations on individual tasks may fail without
 2636 causing the overall batch operation to fail.
 2637

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
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	Add a comment to a task. Returns an identifier that can be used to later update or delete the comment.	In <ul style="list-style-type: none"> task identifier plain text Out <ul style="list-style-type: none"> comment identifier 	No	(any state)	(no state transition)
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
complete	Execution of the task finished successfully. The fault <code>hta:illegalStateFault</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> .	In <ul style="list-style-type: none"> task identifier output data of task (optional) Out <ul style="list-style-type: none"> void 	Yes	InProgress	Completed

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
delegate	Assign the task to one user and set the task to state <i>Reserved</i> . If the recipient was not a potential owner then this person MUST be added to the set of potential owners. For details on delegating human tasks refer to section 4.10.3.	In <ul style="list-style-type: none"> task identifier organizational entity (<code>http:taskOrganizationalEntity</code>) Out <ul style="list-style-type: none"> void 	Yes	Ready Reserved InProgress	Reserved
deleteAttachment	Delete the attachment with the specified identifier from the task. Attachments provided by the enclosing context MUST NOT be affected by this operation.	In <ul style="list-style-type: none"> task identifier attachment identifier Out <ul style="list-style-type: none"> void 	No	(any state)	(no state transition)
deleteComment	Deletes the identified comment.	In <ul style="list-style-type: none"> task identifier comment identifier Out <ul style="list-style-type: none"> void 	No	(any state)	(no state transition)
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)
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)

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
fail	<p>Execution of the task fails and a fault is returned.</p> <p>The fault <code>hta:illegalOperationFault</code> MUST be returned if the task interface defines no faults.</p> <p>The fault <code>hta:illegalStateFault</code> 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 <code>setFault</code>.</p>	<p>In</p> <ul style="list-style-type: none"> task identifier fault (optional) – contains the fault name and fault data <p>Out</p> <ul style="list-style-type: none"> void 	Yes	InProgress	Failed
forward	<p>Forward the task to another organization entity. The WS-HumanTask Client MUST specify the receiving organizational entity.</p> <p>Potential owners MAY forward a task while the task is in the <i>Ready</i> state.</p> <p>For details on forwarding human tasks refer to section 4.10.3.</p>	<p>In</p> <ul style="list-style-type: none"> task identifier organizational entity (<code>htt:tOrganizationalEntity</code>) <p>Out</p> <ul style="list-style-type: none"> void 	Yes	Ready Reserved InProgress	Ready
getAttachment	Get the task attachment with the given identifier.	<p>In</p> <ul style="list-style-type: none"> task identifier attachment identifier <p>Out</p> <ul style="list-style-type: none"> <code>htt:attachment</code> 	No	(any state)	(no state transition)

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
getAttachmentInfos	Get attachment information for all attachments associated with the task.	In <ul style="list-style-type: none"> task identifier Out <ul style="list-style-type: none"> list of attachment data (list of <code>htt:attachmentInfo</code>) 	No	(any state)	(no state transition)
getComments	Get all comments of a task	In <ul style="list-style-type: none"> task identifier Out <ul style="list-style-type: none"> list of comments (list of <code>htt:comment</code>) 	No	(any state)	(no state transition)
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)
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)
getOutcome	Get the outcome of the task	In <ul style="list-style-type: none"> task identifier Out <ul style="list-style-type: none"> string 	No	(any state)	(no state transition)

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
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)
getParentTask	Returns the superior composite task of a sub task	In <ul style="list-style-type: none"> task identifier Out <ul style="list-style-type: none"> htt:tTaskDetails 	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)
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	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)
getSubtasks	Returns all sub tasks of a task (created instances)	In <ul style="list-style-type: none"> task identifier Out <ul style="list-style-type: none"> list of tasks (list of htt:tTaskDetails) 	No	(any state)	(no state transition)

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
getTaskDescription	Applies to both tasks and notifications. Returns the presentation description in the specified mime type.	In <ul style="list-style-type: none"> task identifier content type – optional, default is text/plain Out <ul style="list-style-type: none"> string 	No	(any state)	(no state transition)
getTaskDetails	Applies to both tasks and notifications. Returns a data object of type <code>htt:tTaskDetails</code>	In <ul style="list-style-type: none"> task identifier Out <ul style="list-style-type: none"> task (<code>htt:tTaskDetails</code>) 	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 (<code>htt:tTaskHistoryFilter</code>) startIndex maxTasks includeData Out <ul style="list-style-type: none"> list of <code>htt:taskEvent</code> 	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	Applies to tasks. Returns list of operations that are available to the authorized user given the user's role and the state of the task.	In <ul style="list-style-type: none"> task identifier Out <ul style="list-style-type: none"> List of available operation. 	No	(any state)	(no state transition)
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)
instantiateSubTask	Creates an instantiateable subtask for the task from the definition of the task. The fault <code>hta:illegalArgumentFault</code> MUST be returned if the task does not have an instantiateable subtask of the given name. Returns the identifier for the created subtask.	In <ul style="list-style-type: none"> task identifier subtask name Out <ul style="list-style-type: none"> task identifier 	No	Reserved In Progress	(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)
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

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

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)
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 <ul style="list-style-type: none"> 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 <ul style="list-style-type: none"> void 	Yes	Created Ready Reserved In Progress	(no state transition)
setTaskStartDeadlineExpression	Sets a deadline expression for the named start deadline of the task	In <ul style="list-style-type: none"> task identifier deadline name deadline expression Out <ul style="list-style-type: none"> void 	Yes	Created Ready Reserved In Progress	(no state transition)
setTaskStartDurationExpression	Sets a duration expression for the named start deadline of the task	In <ul style="list-style-type: none"> task identifier deadline name duration expression Out <ul style="list-style-type: none"> void 	Yes	Created Ready Reserved In Progress	(no state transition)

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
skip	Skip the task. If the task is not skipable then the fault <code>hta:illegalOperationFault</code> MUST be returned.	In <ul style="list-style-type: none"> task identifier Out <ul style="list-style-type: none"> void 	Yes	Created Ready Reserved InProgress	Obsolete
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
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
updateComment	Updates the identified comment with the supplied new text.	In <ul style="list-style-type: none"> task identifier comment identifier plain text Out <ul style="list-style-type: none"> void 	No	(any state)	(no state transition)

2638

2639 7.1.2 Simple Query Operations

2640 Simple query operations allow retrieving task data. These operations MUST be supported by a WS-
 2641 HumanTask Processor. The identity of the user is implicitly passed when invoking any of the following
 2642 operations.

2643 The following operations will return both matching tasks and sub tasks.

2644

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> ("<="), <i>greater than or equals</i> (">="), and the <i>IN</i> operator for multi-valued user/group elements of generic human roles. An example of a where clause is "task.priority = 1". A value of type <code>xsd:QName</code> MUST be</p>	In <ul style="list-style-type: none"> task type ("ALL" "TASKS" "NOTIFICATIONS") generic human role work queue status list where clause order-by clause created-on clause maxTasks taskIndexOffset Out <ul style="list-style-type: none"> list of tasks (list of <code>htt:tTaskAbstract</code>) 	Any

Operation Name	Description	Parameters	Authorization
	<p>specified as string in the format "{namespaceURI}localName", where the {namespace} part is optional and treated as wildcard if not specified. An example using a QName is "task.name = '{http://example.com}ApproveClaim'". A comparison with a value of type <code>htt:OrganizationalEntity</code> MUST be performed using its user/group child elements. An example is "task.potentialOwner.user IN ('Joe', 'Fred') OR task.potentialOwner.group = 'approvers'".</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 <code>Task.CreatedTime</code> with operators as described above. The combination of the two clauses enables simple but restricted paging in a task list client.</p> <p>If <code>maxTasks</code> is specified, then the number of task abstracts returned for this query MUST NOT exceed this limit. The <code>taskIndexOffset</code> can be used to perform multiple identical queries and iterate over result sets where the <code>maxTasks</code> size exceeds the query limit. If <code>maxTasks</code> 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 tasks in all valid states are returned.</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>The where clause is optional. If specified, it MUST follow the same rules described for the <code>getMyTaskAbstracts</code> operation.</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 <code>Task.CreatedTime</code> with operators as described above.</p> <p>The combination of the two clauses enables simple but restricted paging in the task list client.</p> <p>If <code>maxTasks</code> is specified, then the number of task details returned for this query MUST NOT exceed this limit. If <code>maxTasks</code> has not been specified then all tasks fulfilling the query are returned.</p>		

2645

2646 The return types `tTaskAbstract` and `tTaskDetails` are defined in section 3.8.4 “Data Types for Task
 2647 Instance Data”.

2648 **Simple Task View**

2649 The table below lists the task attributes available to the simple query operations. This view is used when
 2650 defining the where clause of any of the above query operations.

2651

Column Name	Type
ID	<code>xsd:anyURI</code>
TaskType	Enumeration
Name	<code>xsd:QName</code>
Status	Enumeration (for values see 4.10 “Human Task Behavior and State Transitions”)
Priority	<code>htt:tPriority</code>
CreatedTime	<code>xsd:dateTime</code>
ActivationTime	<code>xsd:dateTime</code>
ExpirationTime	<code>xsd:dateTime</code>
HasPotentialOwners	<code>xsd:boolean</code>

Column Name	Type
StartByTimeExists	xsd:boolean
CompleteByTimeExists	xsd:boolean
RenderingMethodExists	xsd:boolean
Escalated	xsd:boolean
ParentTaskId	xsd:anyURI
HasSubTasks	xsd:boolean
SearchBy	xsd:string
Outcome	xsd:string

2652

2653 7.1.3 Advanced Query Operation

2654 The advanced query operation is used by the task list client to perform queries not covered by the simple
 2655 query operations defined in 7.1.2. A WS-HumanTask Processor MAY support this operation. An
 2656 implementation MAY restrict the results according to authorization of the invoking user.

2657

2658 The following operations will return both matching tasks and sub tasks.

2659

Operation Name	Description	Parameters
query	<p>Retrieve task data. All clauses assume a (pseudo-) SQL syntax. If maxTasks is specified, then the number of task returned by the query MUST NOT exceed this limit. The taskIndexOffset can be used to perform multiple identical queries and iterate over result sets where the maxTasks size exceeds the query limit.</p> <p>For data of type <code>xsd:QName</code> or <code>htt:tOrganizationalEntity</code> in a where clause, see the description of the <code>getMyTaskAbstracts</code> operation in section 7.1.2.</p>	<p>In</p> <ul style="list-style-type: none"> • select clause • where clause • order-by clause • maxTasks • taskIndexOffset <p>Out</p> <ul style="list-style-type: none"> • task query result set (<code>htt:tTaskQueryResultSet</code>)

2660

2661 ResultSet Data Type

2662 This is the result set element that is returned by the `query` operation.

```
2663 <xsd:element name="taskQueryResultSet" type="tTaskQueryResultSet" />
2664 <xsd:complexType name="tTaskQueryResultSet">
2665   <xsd:sequence>
```

```

2666     <xsd:element name="row" type="tTaskQueryResultRow"
2667               minOccurs="0" maxOccurs="unbounded" />
2668   </xsd:sequence>
2669 </xsd:complexType>
2670

```

2671 The following is the type of the row element contained in the result set. The value in the row are returned
 2672 in the same order as specified in the select clause of the query.

```

2673 <xsd:complexType name="tTaskQueryResultRow">
2674   <xsd:choice minOccurs="0" maxOccurs="unbounded">
2675     <xsd:element name="id" type="xsd:anyURI"/>
2676     <xsd:element name="taskType" type="xsd:string"/>
2677     <xsd:element name="name" type="xsd:QName"/>
2678     <xsd:element name="status" type="tStatus"/>
2679     <xsd:element name="priority" type="tPriority"/>
2680     <xsd:element name="taskInitiator"
2681                 type="tUser"/>
2682     <xsd:element name="taskStakeholders"
2683                 type="tOrganizationalEntity"/>
2684     <xsd:element name="potentialOwners"
2685                 type="tOrganizationalEntity"/>
2686     <xsd:element name="businessAdministrators"
2687                 type="tOrganizationalEntity"/>
2688     <xsd:element name="actualOwner" type="tUser"/>
2689     <xsd:element name="notificationRecipients"
2690                 type="tOrganizationalEntity"/>
2691     <xsd:element name="createdTime" type="xsd:dateTime"/>
2692     <xsd:element name="createdBy" type="tUser"/>
2693     <xsd:element name="lastModifiedTime" type="xsd:dateTime"/>
2694     <xsd:element name="lastModifiedBy" type="tUser"/>
2695     <xsd:element name="activationTime" type="xsd:dateTime"/>
2696     <xsd:element name="expirationTime" type="xsd:dateTime"/>
2697     <xsd:element name="isSkipable" type="xsd:boolean"/>
2698     <xsd:element name="hasPotentialOwners" type="xsd:boolean"/>
2699     <xsd:element name="startByTime" type="xsd:dateTime"/>
2700     <xsd:element name="completeByTime" type="xsd:dateTime"/>
2701     <xsd:element name="presentationName" type="tPresentationName"/>
2702     <xsd:element name="presentationSubject"
2703                 type="tPresentationSubject"/>
2704     <xsd:element name="renderingMethodName" type="xsd:QName"/>
2705     <xsd:element name="hasOutput" type="xsd:boolean"/>
2706     <xsd:element name="hasFault" type="xsd:boolean"/>
2707     <xsd:element name="hasAttachments" type="xsd:boolean"/>
2708     <xsd:element name="hasComments" type="xsd:boolean"/>
2709     <xsd:element name="escalated" type="xsd:boolean"/>
2710     <xsd:element name="parentTaskId" type="xsd:anyURI"/>
2711     <xsd:element name="hasSubTasks" type="xsd:boolean"/>
2712     <xsd:element name="searchBy" type="xsd:string"/>
2713     <xsd:element name="outcome" type="xsd:string"/>
2714     <xsd:element name="taskOperations" type="tTaskOperations"/>
2715     <xsd:any namespace="##other" processContents="lax"/>
2716   </xsd:choice>
2717 </xsd:complexType>

```

2718
 2719

2720 Complete Task View

2721 The table below is the set of columns used when defining select clause, where clause, and order-by
 2722 clause of query operations. Conceptually, this set of columns defines a universal relation. As a result the
 2723 query can be formulated without specifying a from clause. A WS-HumanTask Processor MAY extend this
 2724 view by adding columns.
 2725

Column Name	Type	Constraints
ID	xsd:anyURI	
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)	htt:tUser or htt:tOrganizationalEntity	
CreatedTime	xsd:dateTime	The time in UTC when the task has been created.
CreatedBy	htt:tUser	
LastModifiedTime	xsd:dateTime	The time in UTC when the task has been last modified.
LastModifiedBy	htt:tUser	
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.

Column Name	Type	Constraints
CompleteByTime	xsd:dateTime	The time in UTC when the task needs to be completed. This time corresponds to the respective end deadline.
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:anyURI	
HasSubTasks	xsd:boolean	
SearchBy	xsd:string	
Outcome	xsd:string	
TaskOperations	htt:tTaskOperations	

2726

2727 **7.1.4 Administrative Operations**

2728 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	In task identifier organizational entity (htt:tOrganizationalEntity)	Yes	Created	Ready Reserved

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
	task is <i>Reserved</i> . If it is nominated to several people then the new state of the task is <i>Ready</i> .	Out void			
setGenericHumanRole	Replace the organizational assignment to the task in one generic human role.	In task identifier generic human role organizational entity (http:OrganizationEntity) Out void	Yes	Created Ready Reserved InProgress Suspended/Ready (from Ready) Suspended/Reserved (from Reserved) Suspended/InProgress (from InProgress)	(no state transition)

2729

2730

2731 7.1.5 Operation Authorizations

2732 The table below summarizes the required authorizations in terms of generic human roles to execute
 2733 participant, query and administrative operations. Thus, it is a precise definition of the generic human roles
 2734 as well. The sign plus ('+') means that the operation MUST be available for the generic human role. The
 2735 sign minus ('-') means that the operation MUST NOT be available for the generic human role. 'n/a'
 2736 indicates that the operation is not applicable and thus MUST NOT be available for the generic human
 2737 role. 'MAY' defines that vendor MAY chose to support the operation for the generic human role.

2738 If a person has multiple generic human roles on a human task or notification and she is allowed to
 2739 perform an operation in any of the roles then the invocation of the operation will not fail, otherwise
 2740 `hta:illegalAccessFault` and `hta:recipientNotAllowed` MUST be returned in the case of tasks
 2741 and notifications respectively. If a person is included in the list of excluded owners of a task then she
 2742 MUST NOT perform any of the operations.

2743 All batch operations (operations with a name prefix "batch") may be invoked by any caller; no specific
 2744 authorization is required. Missing authorizations for operations on individual tasks result in a report entry
 2745 in the batch operation's response message.

2746

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

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
suspend	MAY	+	MAY	MAY	+	n/a
suspendUntil	MAY	+	MAY	MAY	+	n/a
updateComment	MAY	+	+	+	+	n/a

2748

2749 7.2 XPath Extension Functions

2750 This section introduces XPath extension functions that are provided to be used within the definition of a
 2751 human task or notification. A WS-HumanTask Processor MUST support the XPath Functions listed below.
 2752 When defining properties using these XPath functions, note the initialization order in section 4.10.1.

2753 Definition of these XPath extension functions is provided in the table below. Input parameters that specify
 2754 task name, message part name or logicalPeopleGroup name MUST be literal strings. This restriction
 2755 does not apply to other parameters. Because XPath 1.0 functions do not support returning faults, an
 2756 empty node set is returned in the event of an error.

2757 XPath functions used for notifications in an escalation can access context from the enclosing task by
 2758 specifying that task's name.

Operation Name	Description	Parameters
getActualOwner	Returns the actual owner of the task. It MUST evaluate to an empty <code>htt:user</code> in case there is no actual owner. If the task name is not present the current task MUST be considered.	In <ul style="list-style-type: none"> task name (optional) Out <ul style="list-style-type: none"> the actual owner (user id as <code>htt:user</code>)
getBusinessAdministrators	Returns the business administrators of the task. It MUST evaluate to an empty <code>htt:organizationalEntity</code> in case of an error. If the task name is not present the current task MUST be considered.	In <ul style="list-style-type: none"> task name (optional) Out <ul style="list-style-type: none"> business administrators (<code>htt:organizationalEntity</code>)
getCountOfFinishedSubTasks	Returns the number of finished sub tasks of a task If the task name is not present the current task MUST be considered	In <ul style="list-style-type: none"> task name (optional) Out <ul style="list-style-type: none"> Number of the finished task sub-tasks. If the task doesn't have sub tasks then 0 is returned
getCountOfSubTasks	Returns the number of sub tasks of a task If the task name is not present the current task MUST be considered	In <ul style="list-style-type: none"> task name (optional) Out <ul style="list-style-type: none"> Number of the task sub-tasks. If the task doesn't have sub tasks then 0 is returned
getCountOfSubTasksInState	Returns the number of a task subtasks that are in the specified state If the task name is not present the current task MUST be considered	In <ul style="list-style-type: none"> state task name (optional) Out <ul style="list-style-type: none"> Number of the task sub tasks in the specified state. If the task doesn't have sub tasks then 0 is returned
getCountOfSubTasksWithOutcome	Returns the number of a task sub tasks that match the given outcome If the task name is not present the current task	In <ul style="list-style-type: none"> outcome task name (optional) Out

Operation Name	Description	Parameters
	MUST be considered	<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 <code>htt:organizationalEntity</code> 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 (<code>htt:organizationalEntity</code>)
getInput	<p>Returns the part of the task's input message.</p> <p>If the task name is not present the current task MUST be considered.</p>	<p>In</p> <ul style="list-style-type: none"> part name task name (optional) <p>Out</p> <ul style="list-style-type: none"> input message part
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>htt: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>htt:organizationalEntity</code>)
getOutcome	<p>Returns the outcome of the task. It MUST evaluate to an empty string in case there is no outcome specified for the task.</p> <p>If the task name is not present the current task MUST be considered.</p>	<p>In</p> <ul style="list-style-type: none"> task name (optional) <p>Out</p> <ul style="list-style-type: none"> the task outcome (<code>xsd:string</code>)

Operation Name	Description	Parameters
getOutput	Returns the part of the task's output message. 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 part
getPotentialOwners	Returns the potential owners of the task. It MUST evaluate to an empty <code>htt:organizationalEntity</code> in case of an error. If the task name is not present the current task MUST be considered.	In <ul style="list-style-type: none"> task name (optional) Out <ul style="list-style-type: none"> potential owners (<code>htt:organizationalEntity</code>)
getSubtaskOutput	Returns a node-set representing the specified part or contained elements of a sub task's output message. Only completed sub tasks of the current task MUST be considered	In <ul style="list-style-type: none"> sub task name part name location path 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
getTaskInitiator	Returns the initiator of the task. It MUST evaluate to an empty <code>htt:user</code> in case there is no initiator. If the task name is not present the current task MUST be considered.	In <ul style="list-style-type: none"> task name (optional) Out <ul style="list-style-type: none"> the task initiator (user id as <code>htt:user</code>)
getTaskPriority	Returns the priority of the task. It MUST evaluate to "5" in case the priority is not explicitly set.	In <ul style="list-style-type: none"> task name (optional) Out <ul style="list-style-type: none"> priority (<code>htt:tPriority</code>)

Operation Name	Description	Parameters
	If the task name is not present the current task MUST be considered.	
getTaskStakeholders	Returns the stakeholders of the task. It MUST evaluate to an empty <code>htt:organizationalEntity</code> in case of an error. If the task name is not present the current task MUST be considered.	In <ul style="list-style-type: none"> task name (optional) Out <ul style="list-style-type: none"> task stakeholders (<code>htt:organizationalEntity</code>)

2760

2761 Generic set functions:

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

	occurs in either set1 or set2 , eliminating duplicate users.	<pre> ty htt:user) • set2 (htt:organizationalEnti ty htt:user) Out • result (htt:organizationalEnti ty) </pre>
--	---------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------

2762

2763 In addition to the general-purpose functions listed above, the following aggregation functions MUST be
2764 supported by a WS-HumanTask Processor. All aggregation functions take a node-set of strings,
2765 booleans, or numbers as the first input parameter, and produce a result of the same type.

2766 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 empty node-set)	In <ul style="list-style-type: none"> node-set of string nodes percentage

2767

2768

2769

2770

2771

2772 Boolean-valued aggregation functions:

Operation Name	Description	Parameters
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

2773

2774 Number-valued aggregation functions:

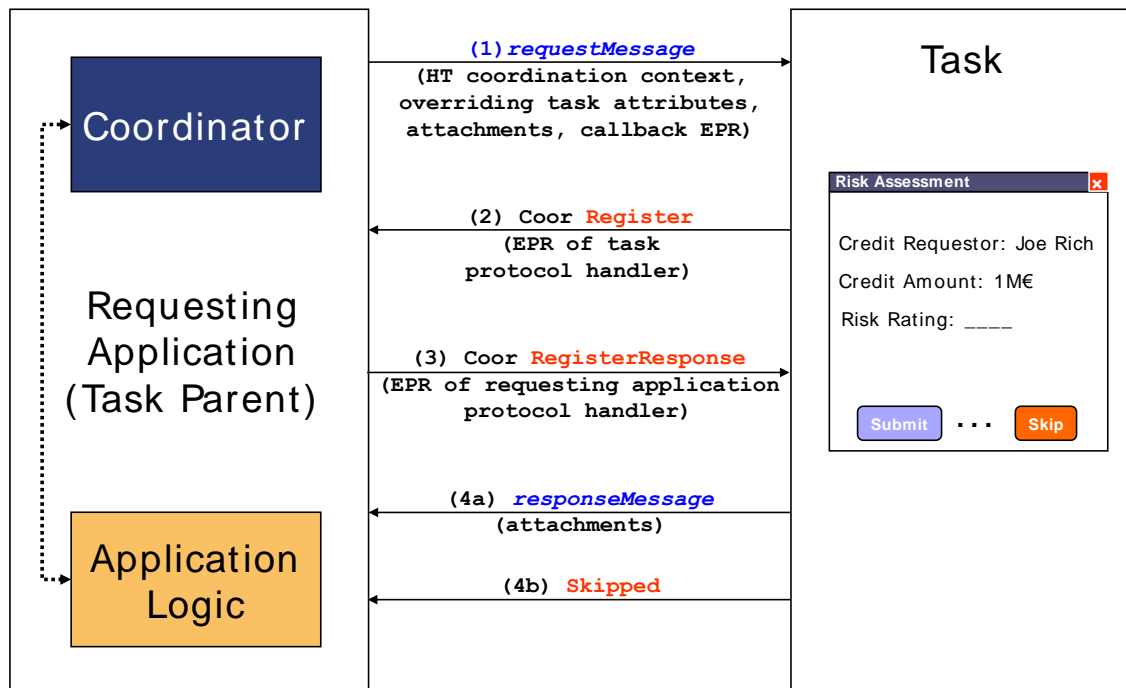
Operation Name	Description	Parameters
avg	Returns the average value of all number nodes - returns NaN for an empty node-set	In <ul style="list-style-type: none"> node-set of number nodes
max	Returns the maximum value of all number nodes - returns NaN for an empty node-set	In <ul style="list-style-type: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

2775

2776
2777
2778
2779
2780
2781
2782
2783
2784

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.



2785

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

2786 While we do not make any assumptions about the nature of the application in the following scenarios, in
2787 practice it would be hosted by an infrastructure that actually deals with the WS-HumanTask coordination
2788 protocol on the application's behalf.

2789 In case of human tasks the following message exchanges are possible.

2790 **Scenario 1:** At some point in time, the application invokes the human task through its service interface. In
2791 order to signal to the WS-HumanTask Processor that an instance of the human task can be created
2792 which is actually coordinated by the parent application, this request message contains certain control
2793 information. This control information consists of a coordination context of the WS-HumanTask
2794 coordination protocol, and optional human task attributes that are used to override aspects of the human
2795 task definition.

- 2796 • The coordination context (see [WS-C] for more details on Web services coordination framework
2797 used here) contains the element `CoordinationType` that MUST specify the WS-HumanTask
2798 coordination type [http://docs.oasis-open.org/ns/bpel4people/ws-](http://docs.oasis-open.org/ns/bpel4people/ws-humantask/protocol/200803)
2799 [humantask/protocol/200803](http://docs.oasis-open.org/ns/bpel4people/ws-humantask/protocol/200803). The inclusion of a coordination context within the request

2800 message indicates that the life cycle of the human tasks is managed via corresponding protocol
2801 messages from outside the WS-HumanTask Processor. The coordination context further contains
2802 in its `RegistrationService` element an endpoint reference that the WS-HumanTask
2803 Processor MUST use to register the task as a participant of that coordination type.
2804 Note: In a typical implementation, the parent application or its environment will create that
2805 coordination context by issuing an appropriate request against the WS-Coordination (WS-C)
2806 activation service, followed by registering the parent application as a `TaskParent` participant in
2807 that protocol.

- 2808 • The optional human task attributes allow overriding aspects of the definition of the human task
2809 from the calling application. The WS-HumanTask Parent MAY set values of the following
2810 attributes of the task definition:
 - 2811 ○ Priority of the task
 - 2812 ○ Actual people assignments for each of the generic human roles of the human task
 - 2813 ○ The skipable indicator which determines whether a task can actually be skipped at
2814 runtime.
 - 2815 ○ The amount of time by which the task activation is deferred.
 - 2816 ○ The expiration time for the human task after which the calling application is no longer
2817 interested in its result.

2818 After having created this request message, it is sent to the WS-HumanTask Processor (step (1) in Figure
2819 10). The WS-HumanTask Processor receiving that message MUST extract the coordination context and
2820 callback information, the human task attributes (if present) and the application payload. Before applying
2821 this application payload to the new human task, the WS-HumanTask Processor MUST register the human
2822 task to be created with the registration service passed as part of the coordination context (step (2) in
2823 Figure 10). The corresponding WS-C `Register` message MUST include the endpoint reference (EPR) of
2824 the protocol handler of the WS-HumanTask Processor that the WS-HumanTask Parent MUST use to
2825 send all protocol messages to WS-HumanTask Processor. This EPR is the value contained in the
2826 `ParticipantProtocolService` element of the `Register` message. Furthermore, the registration
2827 MUST be as a `HumanTask` participant by specifying the corresponding value in the
2828 `ProtocolIdentifier` element of the `Register` message. The WS-HumanTask Parent reacts to that
2829 message by sending back a `RegisterResponse` message. This message MUST contain in its
2830 `CoordinatorProtocolService` element the EPR of the protocol handler of the parent application,
2831 which MUST be used by the WS-HumanTask Processor for sending protocol messages to the parent
2832 application (step (3) in Figure 10).

2833 Now the instance of the human task is activated by the WS-HumanTask Processor, so the assigned
2834 person can perform the task (e.g. the risk assessment). Once the human task is successfully completed,
2835 a response message MUST be passed back to the parent application (step (4a) in Figure 10) by WS-
2836 HumanTask Processor.

2837 **Scenario 2:** If the human task is not completed with a result, but the assigned person determines that the
2838 task can be skipped (and hence reaches its *Obsolete* final state), then a “skipped” coordination protocol
2839 message MUST be sent from the WS-HumanTask Processor to its parent application (step (4b) in Figure
2840 10). No response message is passed back.

2841 **Scenario 3:** If the WS-HumanTask Parent needs to end prematurely before the invoked human task has
2842 been completed, it MUST send an `exit` coordination protocol message to the WS-HumanTask
2843 Processor causing the WS-HumanTask Processor to end its processing. A response message SHOULD
2844 NOT be passed back by WS-HumanTask Processor.

2845 In case of notifications to WS-HumanTask Processor, only some of the overriding attributes are
2846 propagated with the request message. Only priority and people assignments MAY be overridden for a
2847 notification, and the elements `isSkipable`, `expirationTime` and `attachments` MUST be ignored if present by
2848 WS-HumanTask Processor. Likewise, the WS-HumanTask coordination context, `attachments` and the
2849 callback EPR do not apply to notifications and MUST be ignored as well by WS-HumanTask Processor.
2850 Finally, a notification SHOULD NOT return WS-HumanTask coordination protocol messages. There
2851 SHOULD NOT be a message exchange beyond the initiating request message between the WS-
2852 HumanTask Processor and WS-HumanTask Parent.

2853 **8.1 Human Task Coordination Protocol Messages**

2854 The following section describes the behavior of the human task with respect to the protocol messages
 2855 exchanged with its requesting application which is human task aware. In particular, we describe which
 2856 state transitions trigger which protocol message and vice versa. WS-HumanTask Parent MUST support
 2857 WS-HumanTask Coordination protocol messages in addition to application requesting, responding and
 2858 fault messages.

2859 See diagram in section 4.10 “Human Task Behavior and State Transitions”.

- 2860 1. The initiating message containing a WS-HumanTask coordination context is received by the WS-
 2861 HumanTask Processor. This message MAY include ad hoc attachments that are to be made
 2862 available to the WS-HumanTask Processor. A new task is created. As part of the context, an EPR
 2863 of the registration service MUST be passed by WS-HumanTask Parent. This registration service
 2864 MUST be used by the hosting WS-HumanTask Processor to register the protocol handler
 2865 receiving the WS-HumanTask protocol messages sent by the requesting Application. If an error
 2866 occurs during the task instantiation the final state *Error* is reached and protocol message *fault*
 2867 MUST be sent to the requesting application by WS-HumanTask Processor.
- 2868 2. On successful completion of the task an application level response message MUST be sent and
 2869 the task moved to state *Completed*. When this happens, attachments created during the
 2870 processing of the task MAY be added to the response message. Attachments that had been
 2871 passed in the initiating message MUST NOT be returned. The response message outcome
 2872 MUST be set to the outcome of the task.
- 2873 3. On unsuccessful completion (completion with a fault message), an application level fault
 2874 message MUST be sent and the task moved to state *Failed*. When this happens, attachments
 2875 created during the processing of the task MAY be added to the response message. Attachments
 2876 that had been passed in the initiating message MUST NOT be returned.
- 2877 4. If the task experiences a non-recoverable error protocol message *fault* MUST be sent and
 2878 the task moved to state *Error*. Attachments MUST NOT be returned.
- 2879 5. If the task is skipable and is skipped then the WS-HumanTask Processor MUST send the
 2880 protocol message *skipped* and task MUST be moved to state *Obsolete*. Attachments MUST
 2881 NOT be returned.
- 2882 6. On receipt of protocol message *exit* the task MUST be moved to state *Exited*. This indicates
 2883 that the requesting application is no longer interested in any result produced by the task.

2884 The following table summarizes this behavior, the messages sent, and their direction, i.e., whether a
 2885 message is sent from the requesting application to the task (“out” in the column titled Direction) or vice
 2886 versa (“in”).

2887

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

2888 8.2 Protocol Messages

2889 All WS-HumanTask protocol messages have the following type:

```
2890 <xsd:complexType name="tProtocolMsgType">
2891   <xsd:sequence>
2892     <xsd:any namespace="##other" processContents="lax"
2893       minOccurs="0" maxOccurs="unbounded" />
2894   </xsd:sequence>
2895   <xsd:anyAttribute namespace="##other" processContents="lax" />
2896 </xsd:complexType>
```

2897 This message type is extensible and any implementation MAY use this extension mechanism to define
2898 proprietary attributes and content which are out of the scope of this specification.

2899 8.2.1 Protocol Messages Received by a Task Parent

2900 The following is the definition of the `htcp:skipped` message.

```
2901 <xsd:element name="skipped" type="htcp:tProtocolMsgType" />
2902 <wsdl:message name="skipped">
2903   <wsdl:part name="parameters" element="htcp:skipped" />
2904 </wsdl:message>
```

2905 The `htcp:skipped` message is used to inform the task parent (i.e. the requesting application) that the
2906 invoked task has been skipped. The task does not return any result.

2907 The following is the definition of the `htcp:fault` message.

```
2908 <xsd:element name="fault" type="htcp:tProtocolMsgType" />
2909 <wsdl:message name="fault">
2910   <wsdl:part name="parameters" element="htcp:fault" />
2911 </wsdl:message>
```

2912 The `htcp:fault` message is used to inform the task parent that the task has ended abnormally. The
2913 task does not return any result.

2914 8.2.2 Protocol Messages Received by a Task

2915 Upon receipt of the following `htcp:exit` message the task parent informs the task that it is no longer
2916 interested in its results.

```
2917 <xsd:element name="exit" type="htcp:tProtocolMsgType" />
2918 <wsdl:message name="exit">
2919   <wsdl:part name="parameters" element="htcp:exit" />
2920 </wsdl:message>
```

2921 8.3 WSDL of the Protocol Endpoints

2922 Protocol messages are received by protocol participants via operations of dedicated ports called protocol
2923 endpoints. In this section we specify the WSDL port types of the protocol endpoints needed to run the
2924 WS-HumanTask coordination protocol.

2925 8.3.1 Protocol Endpoint of the Task Parent

2926 An application that wants to create a task and wants to become a task parent MUST provide an endpoint
2927 implementing the following port type. This endpoint is the protocol endpoint of the task parent receiving
2928 protocol messages of the WS-HumanTask coordination protocol from a task. The operation used by the
2929 task to send a certain protocol message to the task parent is named by the message name of the protocol
2930 message concatenated by the string `Operation`. For example, the `skipped` message MUST be passed
2931 to the task parent by using the operation named `skippedOperation`.

```
2932 <wsdl:portType name="clientParticipantPortType">
```

```

2933 <wsdl:operation name="skippedOperation">
2934   <wsdl:input message="htcp:skipped" />
2935 </wsdl:operation>
2936 <wsdl:operation name="faultOperation">
2937   <wsdl:input message="htcp:fault" />
2938 </wsdl:operation>
2939 </wsdl:portType>

```

2940 8.3.2 Protocol Endpoint of the Task

2941 For a WS-HumanTask Definition a task MUST provide an endpoint implementing the following port type.
 2942 This endpoint is the protocol endpoint of the task receiving protocol messages of the WS-HumanTask
 2943 coordination protocol from a task parent. The operation used by the task parent to send a certain protocol
 2944 message to a task is named by the message name of the protocol message concatenated by the string
 2945 Operation. For example, the `exit` protocol message MUST be passed to the task by using the
 2946 operation named `exitOperation`.

```

2947 <wsdl:portType name="humanTaskParticipantPortType">
2948   <wsdl:operation name="exitOperation">
2949     <wsdl:input message="htcp:exit" />
2950   </wsdl:operation>
2951 </wsdl:portType>

```

2952 8.4 Providing Human Task Context

2953 The task context information is exchanged between the requesting application and a task or a notification.
 2954 In case of tasks, this information is passed as header fields of the request and response messages of the
 2955 task's operation. In case of notifications, this information is passed as header fields of the request
 2956 message of the notification's operation.

2957 8.4.1 SOAP Binding of Human Task Context

2958 In general, a SOAP binding specifies for message header fields how they are bound to SOAP headers. In
 2959 case of WS-HumanTask, the `humanTaskRequestContext` and `humanTaskResponseContext`
 2960 elements are simply mapped to SOAP header as a whole. The following listings show the SOAP binding
 2961 of the human task request context and human task response context in an infoset representation.

```

2962 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
2963   xmlns:htc="http://docs.oasis-open.org/ns/bpel4people/ws-
2964   humantask/context/200803">
2965   <S:Header>
2966     <htc:humanTaskRequestContext>
2967       <htc:priority>...</htc:priority>?
2968       <htc:attachments>...</htc:attachments>?
2969       <htc:peopleAssignments>...</htc:peopleAssignments>?
2970       <htc:isSkipable>...</htc:isSkipable>?
2971       <htc:activationDeferralTime>...</htc:activationDeferralTime>?
2972       <htc:expirationTime>...</htc:expirationTime>?
2973       ... extension elements ...
2974     </htc:humanTaskRequestContext>
2975   </S:Header>
2976   <S:Body>
2977     ...
2978   </S:Body>
2979 </S:Envelope>

```

```

2980
2981 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"

```

```

2982         xmlns:htc="http://docs.oasis-open.org/ns/bpel4people/ws-
2983 humantask/context/200803">
2984     <S:Header>
2985         <htc:humanTaskResponseContext>
2986             <htc:priority>...</htc:priority>?
2987             <htc:attachments>...</htc:attachments>?
2988             <htc:actualOwner>...</htc:actualOwner>?
2989             <htc:actualPeopleAssignments>...</htc:actualPeopleAssignments>?
2990             <htc:outcome>...</htc:outcome>?
2991             ... extension elements ...
2992         </htc:humanTaskResponseContext>
2993     </S:Header>
2994     <S:Body>
2995         ...
2996     </S:Body>
2997 </S:Envelope>

```

2998 The following listing is an example of a SOAP message containing a human task request context.

```

2999 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
3000           xmlns:htc="http://docs.oasis-open.org/ns/bpel4people/ws-
3001 humantask/context/200803">
3002     <S:Header>
3003         <htc:humanTaskRequestContext>
3004             <htc:priority>0</htc:priority>
3005             <htc:peopleAssignments>
3006                 <htc:potentialOwners>
3007                     <htt:organizationalEntity>
3008                         <htt:user>Alan</htt:user>
3009                         <htt:user>Dieter</htt:user>
3010                         <htt:user>Frank</htt:user>
3011                         <htt:user>Gerhard</htt:user>
3012                         <htt:user>Ivana</htt:user>
3013                         <htt:user>Karsten</htt:user>
3014                         <htt:user>Matthias</htt:user>
3015                         <htt:user>Patrick</htt:user>
3016                     </htt:organizationalEntity>
3017                 </htc:potentialOwners>
3018             </htc:peopleAssignments>
3019         </htc:humanTaskRequestContext>
3020     </S:Header>
3021     <S:Body>...</S:Body>
3022 </S:Envelope>

```

3023 8.4.2 Overriding Task Definition People Assignments

3024 The task context information exchanged contains a `potentialOwners` element, which can be used at
3025 task creation time to override the set of task assignments that we defined in the original task definition.
3026 Compliant implementations MUST allow overriding of simple tasks and routing patterns that are a single-
3027 level deep, i.e. routing patterns that don't have nested routing patterns. If the task context
3028 `potentialOwners` contains a list of `htt:user` and `htt:group`, and the task definition contains a
3029 routing pattern element `htt:parallel` or `htt:sequence` that has as its only children `htt:user` and
3030 `htt:group` elements, the WS-HumanTask Processor MUST replace the list in the task definition with the
3031 list in the task context. If the task definition contains only a list of `htt:user` and `htt:group`, then the
3032 WS-HumanTask Processor MUST replace the list of users from the task definition with the list of users in
3033 the task context.

3034 8.5 Human Task Policy Assertion

3035 In order to support discovery of Web services that support the human task contract that are available for
3036 coordination by another service, a *human task policy* assertion is defined by WS-HumanTask. This policy
3037 assertion can be associated with the business operation used by the invoking component (recall that the
3038 human task is restricted to have exactly one business operation). In doing so, the provider of a human
3039 task can signal whether or not the corresponding task can communicate with an invoking component via
3040 the WS-HumanTask coordination protocol.

3041 The following describes the policy assertion used to specify that an operation can be used to instantiate a
3042 human task with the proper protocol in place:

```
3043 <http:HumanTaskAssertion wsp:Optional="true"? ...>  
3044 ...  
3045 </http:HumanTaskAssertion>
```

3046 /http:HumanTaskAssertion

3047 This policy assertion specifies that the WS-HumanTask Parent, in this case the sender, **MUST**
3048 include context information for a human task coordination type passed with the message. The
3049 receiving human task **MUST** be instantiated with the WS-Human Task protocol in place by the
3050 WS-HumanTask Processor.

3051 /http:HumanTaskAssertion/@wsp:Optional="true"

3052 As defined in WS-Policy [WS-Policy], this is the compact notation for two policy alternatives, one
3053 with and one without the assertion. Presence of both policy alternatives indicates that the
3054 behavior indicated by the assertion is optional, such that a WS-HumanTask coordination context
3055 **MAY** be passed with an input message. If the context is passed the receiving human task **MUST**
3056 be instantiated with the WS-HumanTask protocol in place. The absence of the assertion is
3057 interpreted to mean that a WS-HumanTask coordination context **SHOULD NOT** be passed with
3058 an input message.

3059 The human task policy assertion indicates behavior for a single operation, thus the assertion has an
3060 Operation Policy Subject. WS-PolicyAttachment [WS-PolAtt] defines two policy attachment points with
3061 Operation Policy Subject, namely wsdl:portType/wsdl:operation and wsdl:binding/wsdl:operation.

3062 The <http:HumanTaskAssertion> policy assertion can also be used for notifications. In that case it
3063 means that the WS-HumanTask Parent, in this case the sender, **MAY** pass the human task context
3064 information with the message. Other headers, including headers with the coordination context are
3065 ignored.

3066 9 Task Parent Interactions with Lean Tasks

3067 9.1 Operations for Task Parent Applications

3068 A number of operations are involved in the life cycle of a lean task definition. These comprise:

- 3069 • Registering a lean task definition, such that it is available for later use
- 3070 • Unregistering a lean task definition, such that it is no longer available for later use
- 3071 • Listing lean task definitions, to determine what is available for use
- 3072 • Creating a lean task from a lean task definition

3073 An operation takes a well-defined set of parameters as its input. Passing an illegal parameter or an illegal
3074 number of parameters MUST result in the `htlt:illegalArgumentFault` being returned. Invoking an
3075 operation that is not allowed in the current state of the lean task definition MUST result in an
3076 `htlt:illegalStateFault`.

3077 By default, the identity of the person on behalf of which the operation is invoked is passed to the WS-
3078 HumanTask Processor. When the person is not authorized to perform the operation the
3079 `htlt:illegalAccessFault` MUST be returned.

3080 This specification does not stipulate the authentication, addressing, and binding scheme employed when
3081 calling an operation. This can be achieved using different mechanisms (e.g. WS-Security, WS-
3082 Addressing).

3083 9.2 Lean Task Interactions

3084 To enable lightweight task definition and creation by a WS-HumanTask Parent, a conformant WS-
3085 HumanTask Processor MUST provide the following operations:

- 3086 • `registerLeanTaskDefinition` API for registration
- 3087 • `unregisterLeanTaskDefinition` API for retraction
- 3088 • `listLeanTaskDefinitions` API for enumeration
- 3089 • `createLeanTask` and `createLeanTaskAsync` APIs for creation

3090 and invoke the following callback operation in response to `createLeanTaskAsync`:

- 3091 • `createLeanTaskAsyncCallback`

3092 9.2.1 Register a Lean Task Definition

```
3093 <xsd:element name="registerLeanTaskDefinition">
3094   <xsd:complexType>
3095     <xsd:sequence>
3096       <xsd:element name="taskDefinition" type="htd:tLeanTask" />
3097     </xsd:sequence>
3098   </xsd:complexType>
3099 </xsd:element>
3100 <xsd:element name="registerLeanTaskDefinitionResponse">
3101   <xsd:complexType>
3102     <xsd:sequence>
3103       <xsd:element name="taskName" type="xsd:NCName" />
3104     </xsd:sequence>
3105   </xsd:complexType>
3106 </xsd:element>
```

3107 The `htlt:registerLeanTaskDefinition` operation is used to create a new Lean Task definition that
3108 is available for future listing and consumption by the `htlt:listLeanTaskDefinitions` and
3109 `htlt:createLeanTask/htlt:createLeanTaskAsync` operations. If an existing Lean Task exists at
3110 the same name as the `htd:tLeanTask/@Name`, the WSHumanTask Processor SHOULD return an
3111 `htlt:illegalStateFault`.

3112 9.2.2 Unregister a Lean Task Definition

```
3113 <xsd:element name="unregisterLeanTaskDefinition">  
3114   <xsd:complexType>  
3115     <xsd:sequence>  
3116       <xsd:element name="taskName" type="xsd:NCName" />  
3117     </xsd:sequence>  
3118   </xsd:complexType>  
3119 </xsd:element>  
3120 <xsd:element name="unregisterLeanTaskDefinitionResponse">  
3121   <xsd:complexType>  
3122     <xsd:sequence>  
3123       <xsd:element name="taskName" type="xsd:NCName" />  
3124     </xsd:sequence>  
3125   </xsd:complexType>  
3126 </xsd:element>
```

3127 The `htlt:unregisterLeanTaskDefinition` operation is used to remove a Lean Task available for
3128 future listing and consumption by the `htlt:listLeanTaskDefinitions` and
3129 `htlt:createLeanTask/htlt:createLeanTaskAsync` operations. The WS-HumanTask Processor
3130 SHOULD also move any instances of lean tasks of this task definition to "Error" state. If the Lean Task
3131 does not already exist as a registered element, the WS-HumanTask Processor MUST return an
3132 `htlt:illegalArgumentFault`.

3133 9.2.3 List Lean Task Definitions

```
3134 <xsd:element name="listLeanTaskDefinitions">  
3135   <xsd:complexType>  
3136     <xsd:sequence>  
3137       <xsd:annotation>  
3138         <xsd:documentation>Empty message</xsd:documentation>  
3139       </xsd:annotation>  
3140     </xsd:sequence>  
3141   </xsd:complexType>  
3142 </xsd:element>  
3143 <xsd:element name="listLeanTaskDefinitionsResponse">  
3144   <xsd:complexType>  
3145     <xsd:sequence>  
3146       <xsd:element name="leanTaskDefinitions">  
3147         <xsd:complexType>  
3148           <xsd:sequence>  
3149             <xsd:element name="leanTaskDefinition" type="htd:tLeanTask"  
3150 minOccurs="0" maxOccurs="unbounded" />  
3151           </xsd:sequence>  
3152         </xsd:complexType>  
3153       </xsd:element>  
3154     </xsd:sequence>  
3155   </xsd:complexType>  
3156 </xsd:element>
```


3157 The `htlt:listLeanTaskDefinitions` operation is used to query the list of `htd:tLeanTask`
3158 elements that are registered Lean Tasks, as registered by the `htlt:registerLeanTaskDefinition`
3159 operation, and not subsequently unregistered by `htlt:unregisterLeanTaskDefinition`.

3160 9.2.4 Create a Lean Task

```
3161 <xsd:element name="CreateLeanTask">  
3162   <xsd:complexType>  
3163     <xsd:sequence>  
3164       <xsd:element name="inputMessage">  
3165         <xsd:complexType>  
3166           <xsd:sequence>  
3167             <xsd:any processContents="lax" namespace="##any" />  
3168           </xsd:sequence>  
3169         </xsd:complexType>  
3170       </xsd:element>  
3171       <xsd:element name="taskDefinition" type="htd:tLeanTask" minOccurs="0"/>  
3172       <xsd:element name="taskName" type="xsd:NCName" minOccurs="0" />  
3173     </xsd:sequence>  
3174   </xsd:complexType>  
3175 </xsd:element>  
3176 <xsd:element name="CreateLeanTaskResponse">  
3177   <xsd:complexType>  
3178     <xsd:sequence>  
3179       <xsd:element name="outputMessage">  
3180         <xsd:complexType>  
3181           <xsd:sequence>  
3182             <xsd:any processContents="lax" namespace="##any" />  
3183           </xsd:sequence>  
3184         </xsd:complexType>  
3185       </xsd:element>  
3186     </xsd:sequence>  
3187   </xsd:complexType>  
3188 </xsd:element>  
3189
```

3190 The `htlt:createLeanTask` operation is called by a WS-HumanTask Parent to create a task based on
3191 a Lean Task definition. This task definition either can be passed in directly to the operation or can
3192 reference a Lean Task definition previously sent via `htlt:registerLeanTaskDefinition`. These
3193 tasks follow the standard pattern of the Human Task Coordination protocol and is the operation on the
3194 `portType` used to create a task in that standard pattern, using the `humanTaskRequestContext` and
3195 `humanTaskResponseContext` as described in section 8.4.

3196 If both `taskName` and `taskDefinition` are set, the WS-HumanTask Processor MUST return an
3197 `htlt:illegalArgumentFault`. If `taskName` is set and a lean task has been registered by that name,
3198 the WS-HumanTask Process MUST use the registered lean task definition to create the task. If `taskName`
3199 is not set and a lean task has not been registered by that name, the WS-HumanTask Processor MUST
3200 return an `htlt:illegalArgumentFault`. If `taskDefinition` is set, the WS-HumanTask Processor MUST
3201 use the `taskDefinition` element as the type of the task to create. The WS-HumanTask Processor MUST
3202 use the `inputMessage` as the input message of the task and return the output message of the task in
3203 the `outputMessage` element.

3204 The `htlt:createLeanTask` operation is long-running because its execution includes the user
3205 interaction with the task owner. As a result, it is not meaningful to bind the request-response operation to
3206 a protocol that blocks any resources until the response is returned.

3207 Alternatively, instead of invoking the long-running request-response operation defined above, an
3208 interaction style using an asynchronous callback operation can be used. In this case, the WS-HumanTask
3209 Parent invokes the following `htlt:createLeanTaskAsync` operation and, as described in section 10,

3210 passes a WS-Addressing endpoint reference (EPR) in order to provide a callback address for delivering
3211 the lean task's output.

3212 Technically, `htlt:createLeanTaskAsync` is also a request-response operation in order to enable
3213 returning faults, but it returns immediately to the caller if the lean task is created successfully, without
3214 waiting for the lean task to complete.

```
3215 <xsd:element name="createLeanTaskAsync">  
3216   <xsd:complexType>  
3217     <xsd:sequence>  
3218       <xsd:element name="inputMessage">  
3219         <xsd:complexType>  
3220           <xsd:sequence>  
3221             <xsd:any processContents="lax" namespace="##any" />  
3222           </xsd:sequence>  
3223         </xsd:complexType>  
3224       </xsd:element>  
3225       <xsd:element name="taskDefinition" type="htd:tLeanTask" minOccurs="0"/>  
3226       <xsd:element name="taskName" type="xsd:NCName" minOccurs="0" />  
3227     </xsd:sequence>  
3228   </xsd:complexType>  
3229 </xsd:element>  
3230 <xsd:element name="createLeanTaskAsyncResponse">  
3231   <xsd:complexType>  
3232     <xsd:sequence/>  
3233   </xsd:complexType>  
3234 </xsd:element>
```

3235 Upon completion of the lean task, the WS-HumanTask Processor invokes the callback operation
3236 `htlt:createLeanTaskAsyncCallback` at the callback address specified in the EPR passed by the
3237 WS-HumanTask Parent.

```
3238 <xsd:element name="createLeanTaskAsyncCallback">  
3239   <xsd:complexType>  
3240     <xsd:sequence>  
3241       <xsd:element name="outputMessage">  
3242         <xsd:complexType>  
3243           <xsd:sequence>  
3244             <xsd:any processContents="lax" namespace="##any" />  
3245           </xsd:sequence>  
3246         </xsd:complexType>  
3247       </xsd:element>  
3248     </xsd:sequence>  
3249   </xsd:complexType>  
3250 </xsd:element>
```

3251 9.2.5 Endpoints for Lean Task Operations

3252 A WS-HumanTask Processor MUST provide an endpoint implementing the following port type. This
3253 endpoint is used to register, unregister, and list lean task definitions, and create a lean task given a
3254 particular definition and input message.

```
3255 <wsdl:portType name="leanTaskOperations">  
3256   <wsdl:operation name="registerLeanTaskDefinition">  
3257     <wsdl:input message="registerLeanTaskDefinition"/>  
3258     <wsdl:output message="registerLeanTaskDefinitionResponse"/>  
3259     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>  
3260     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>  
3261   </wsdl:operation>  
3262 </wsdl:portType>  
3263
```

```

3264 <wsdl:operation name="unregisterLeanTaskDefinition">
3265   <wsdl:input message="unregisterLeanTaskDefinition"/>
3266   <wsdl:output message="unregisterLeanTaskDefinitionResponse"/>
3267   <wsdl:fault name="illegalArgumentFault" message="illegalArgumentFault"/>
3268   <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
3269 </wsdl:operation>
3270
3271 <wsdl:operation name="listLeanTaskDefinitions">
3272   <wsdl:input message="listLeanTaskDefinitions"/>
3273   <wsdl:output message="listLeanTaskDefinitionsResponse"/>
3274   <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
3275 </wsdl:operation>
3276
3277 <wsdl:operation name="createLeanTask">
3278   <wsdl:input message="createLeanTask"/>
3279   <wsdl:output message="createLeanTaskResponse"/>
3280   <wsdl:fault name="illegalArgumentFault" message="illegalArgumentFault"/>
3281   <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
3282 </wsdl:operation>
3283
3284 <wsdl:operation name="createLeanTaskAsync">
3285   <wsdl:input message="createLeanTaskAsync"/>
3286   <wsdl:output message="createLeanTaskAsyncResponse"/>
3287   <wsdl:fault name="illegalArgumentFault" message="illegalArgumentFault"/>
3288   <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
3289 </wsdl:operation>
3290
3291 </wsdl:portType>

```

3292 A WS-HumanTask Parent invoking the `htlt:createLeanTaskAsync` operation MUST provide an
3293 endpoint implementing the following callback port type.

```

3294 <wsdl:portType name="leanTaskCallbackOperations">
3295
3296   <wsdl:operation name="createLeanTaskAsyncCallback">
3297     <wsdl:input message="createLeanTaskAsyncCallback"/>
3298   </wsdl:operation>
3299
3300 </wsdl:portType>

```

3301

10 Providing Callback Information for Human Tasks

3302 WS-HumanTask extends the information model of a WS-Addressing endpoint reference (EPR) defined in
3303 [WS-Addr-Core] (see [WS-Addr-SOAP] and [WS-Addr-WSDL] for more details). This extension is needed
3304 to support passing information to human tasks about ports and operations of a caller receiving responses
3305 from such human tasks.

3306 Passing this callback information from a WS-HumanTask Parent (i.e. a requesting application) to the WS-
3307 HumanTask Processor MAY override static deployment information that may have been set.

3308 10.1 EPR Information Model Extension

3309 Besides the properties of an endpoint reference (EPR) defined by [WS-Addr-Core] WS-HumanTask
3310 defines the following abstract properties:

3311 [response action] : xsd:anyURI (0..1)

3312 This property contains the value of the [action] message addressing property to be sent within the
3313 response message.

3314 [response operation] : xsd:NCName (0..1)

3315 This property contains the name of a WSDL operation.

3316 Each of these properties is a child element of the [metadata] property of an endpoint reference. An
3317 endpoint reference passed by a caller to a WS-HumanTask Processor MUST contain the [metadata]
3318 property. Furthermore, this [metadata] property MUST contain either a [response action] property or a
3319 [response operation] property.

3320 If present, the value of the [response action] property MUST be used by the WS-HumanTask Processor
3321 hosting the responding human task to specify the value of the [action] message addressing property of
3322 the response message sent back to the caller. Furthermore, the [destination] property of this response
3323 message MUST be copied from the [address] property of the EPR contained in the original request
3324 message by the WS-HumanTask Processor.

3325 If present, the value of the [response operation] property MUST be the name of an operation of the port
3326 type implemented by the endpoint denoted by the [address] property of the EPR. The corresponding port
3327 type MUST be included as a WSDL 1.1 definition nested within the [metadata] property of the EPR (see
3328 [WS-Addr-WSDL]). The WS-HumanTask Processor hosting the responding human task MUST use the
3329 value of the [response operation] property as operation of the specified port type at the specified endpoint
3330 to send the response message. Furthermore, the [metadata] property MUST contain WSDL 1.1 binding
3331 information corresponding to the port type implemented by the endpoint denoted by the [address]
3332 property of the EPR.

3333 The EPR sent from the caller to the WS-HumanTask Processor MUST identify the instance of the caller.
3334 This MUST be done by the caller in one of the two ways: First, the value of the [address] property can
3335 contain a URL with appropriate parameters uniquely identifying the caller instance. Second, appropriate
3336 [reference parameters] properties are specified within the EPR. The values of these [reference
3337 parameters] uniquely identify the caller within the scope of the URI passed within the [address] property.

3338 10.2 XML Infoset Representation

3339 The following describes the infoset representation of the EPR extensions introduced by WS-HumanTask:

```
3340 <wsa:EndpointReference>  
3341   <wsa:Address>xsd:anyURI</wsa:Address>  
3342   <wsa:ReferenceParameters>xsd:any*</wsa:ReferenceParameters>?  
3343   <wsa:Metadata>  
3344     <htcp:responseAction>xsd:anyURI</htcp:responseAction>?  
3345     <htcp:responseOperation>xsd:NCName</htcp:responseOperation>?  
3346   </wsa:Metadata>
```

3347 `</wsa:EndpointReference>`

3348 `/wsa:EndpointReference/wsa:Metadata`

3349 This element of the EPR MUST be sent by WS-HumanTask Parent, the caller, to the WS-
3350 HumanTask Processor . It MUST either contain WSDL 1.1 metadata specifying the information to
3351 access the endpoint (i.e. its port type, bindings or ports) according to [WS-Addr-WSDL] as well as
3352 a `<htcp:responseOperation>` element, or it MUST contain a `<htcp:responseAction>`
3353 element.

3354 `/wsa:EndpointReference/wsa:Metadata/htcp:responseAction`

3355 This element (of type `xsd:anyURI`) specifies the value of the [action] message addressing
3356 property to be used by the receiving WS-HumanTask Processor when sending the response
3357 message from the WS-HumanTask Processor back to the caller. If this element is specified the
3358 `<htcp:responseOperation>` element MUST NOT be specified by the caller.

3359 `/wsa:EndpointReference/wsa:Metadata/htcp:responseOperation`

3360 This element (of type `xsd:NCName`) specifies the name of the operation that MUST be used by
3361 the receiving WS-HumanTask Processor to send the response message from the WS-
3362 HumanTask Processor back to the caller.. If this element is specified the
3363 `<htcp:responseAction>` element MUST NOT be specified by the WS-HumanTask Parent.

3364 Effectively, WS-HumanTask defines two ways to pass callback information from the caller to the human
3365 task. First, the EPR contains just the value of the [action] message addressing property that MUST be
3366 used by the WS-HumanTask Processor within the response message (i.e. the
3367 `<htcp:responseAction>` element). Second, the EPR contains the WSDL 1.1 metadata for the port
3368 receiving the response operation. In this case, for the callback information the WS-HumanTask Parent
3369 MUST specify which operation of that port is to be used (i.e. the `<htcp:responseOperation>`
3370 element). In both cases, the response is typically sent to the address specified in the `<wsa:Address>`
3371 element of the EPR contained in the original request message; note, that [WS-Addr-WSDL] does not
3372 exclude redirection to other addresses than the one specified, but the corresponding mechanisms are out
3373 of the scope of the specification.

3374 The following example of an endpoint reference shows the usage of the `<htcp:responseAction>`
3375 element. The `<wsa:Metadata>` elements contain the `<htcp:responseAction>` element that
3376 specifies the value of the [action] message addressing property to be used by the WS-HumanTask
3377 Processor when sending the response message back to the caller. This value is
3378 `http://example.com/LoanApproval/approvalResponse`. The value of the [destination] message
3379 addressing property to be used is given in the `<wsa:Address>` element, namely
3380 `http://example.com/LoanApproval/loan?ID=42`. Note that this URL includes the HTTP search
3381 part with the parameter `ID=42` which uniquely identifies the instance of the caller.

```

3382 <wsa:EndpointReference
3383   xmlns:wsa="http://www.w3.org/2005/08/addressing">
3384
3385   <wsa:Address>http://example.com/LoanApproval/loan?ID=42</wsa:Address>
3386
3387   <wsa:Metadata>
3388     <htcp:responseAction>
3389       http://example.com/LoanApproval/approvalResponse
3390     </htcp:responseAction>
3391   </wsa:Metadata>
3392 </wsa:EndpointReference>

```

3394 The following example of an endpoint reference shows the usage of the `<htcp:responseOperation>`
3395 element and corresponding WSDL 1.1 metadata. The port type of the caller that receives the response
3396 message from the WS-HumanTask Processor is defined using the `<wsdl:portType>` element. In our
3397 example it is the `LoanApprovalPT` port type. The definition of the port type is nested in a corresponding
3398 WSLD 1.1 `<wsdl:definitions>` element in the `<wsa:Metadata>` element. This

3399 <wsdl:definitions> element also contains a binding for this port type as well as a corresponding
3400 port definition nested in a <wsdl:service> element. The <http:responseOperation> element
3401 specifies that the approvalResponse operation of the LoanApprovalPT port type is used to send the
3402 response to the caller. The address of the actual port to be used which implements the
3403 LoanApprovalPT port type and thus the approvalResponse operation is given in the
3404 <wsa:Address> element, namely the URL http://example.com/LoanApproval/loan. The
3405 unique identifier of the instance of the caller is specified in the <xmp:MyInstanceID> element nested in
3406 the <wsa:ReferenceParameters> element.

```
3407 <wsa:EndpointReference
3408   xmlns:wsa="http://www.w3.org/2005/08/addressing">
3409
3410   <wsa:Address>http://example.com/LoanApproval/loan</wsa:Address>
3411
3412   <wsa:ReferenceParameters>
3413     <xmp:MyInstanceID>42</xmp:MyInstanceID>
3414   </wsa:ReferenceParameters>
3415
3416   <wsa:Metadata>
3417
3418     <wsdl:definitions ...>
3419
3420       <wsdl:portType name="LoanApprovalPT">
3421         <wsdl:operation name="approvalResponse">...</wsdl:operation>
3422         ...
3423       </wsdl:portType>
3424
3425       <wsdl:binding name="LoanApprovalSoap" type="LoanApprovalPT">
3426         ...
3427       </wsdl:binding>
3428
3429       <wsdl:service name="LoanApprovalService">
3430         <wsdl:port name="LA" binding="LoanApprovalSoap">
3431           <soap:address
3432             location="http://example.com/LoanApproval/loan" />
3433         </wsdl:port>
3434         ...
3435       </wsdl:service>
3436
3437     </wsdl:definitions>
3438
3439     <http:responseOperation>approvalResponse</http:responseOperation>
3440
3441   </wsa:Metadata>
3442 </wsa:EndpointReference>
```

3444 10.3 Message Addressing Properties

3445 Message addressing properties provide references for the endpoints involved in an interaction at the
3446 message level. For this case, WS-HumanTask Processor uses the message addressing properties
3447 defined in [WS-Addr-Core] for the request message as well as for the response message.

3448 The request message sent by the caller (i.e. the requesting application) to the human task uses the
3449 message addressing properties as described in [WS-Addr-Core]. WS-HumanTask refines the use of the
3450 following message addressing properties:

- 3451 • The [reply endpoint] message addressing property MUST contain the EPR to be used by the WS-
3452 HumanTask Processor to send its response to.

- 3453 Note that the [fault endpoint] property MUST NOT be used by WS-HumanTask Processor. This is
 3454 because via one-way operation no application level faults are returned to the caller.
- 3455 The response message sent by the WS-HumanTask Processor to the caller uses the message
 3456 addressing properties as defined in [WS-Addr-Core] and refines the use of the following properties:
- 3457
- The value of the [action] message addressing property is set as follows:
 - 3458 • If the original request message contains the `<htcp:responseAction>` element in the
 3459 `<wsa:Metadata>` element of the EPR of the [reply endpoint] message addressing property,
 3460 the value of the former element MUST be copied into the [action] property of the response
 3461 message by WS-HumanTask Processor.
 - 3462 • If the original request message contains the `<htcp:responseOperation>` element (and,
 3463 thus, WSDL 1.1 metadata) in the `<wsa:Metadata>` element of the EPR of the [reply
 3464 endpoint] message addressing property, the value of the [action] message addressing
 3465 property of the response message is determined as follows:
 - 3466 • Assume that the WSDL 1.1 metadata specifies within the binding chosen a value for the
 3467 `soapaction` attribute on the `soap:operation` element of the response operation.
 3468 Then, this value MUST be used as value of the [action] property by WS-HumanTask
 3469 Processor.
 - 3470 • If no such `soapaction` attribute is provided, the value of the [action] property MUST be
 3471 derived as specified in [WS-Addr-WSDL] by WS-HumanTask Processor.
 - 3472 • Reference parameters are mapped as specified in [WS-Addr-SOAP].

3473 10.4 SOAP Binding

3474 A SOAP binding specifies how abstract message addressing properties are bound to SOAP headers. In
 3475 this case, WS-HumanTask Processor MUST use the mappings as specified by [WS-Addr-SOAP].

3476 The following is an example of a request message sent from the caller to the WS-HumanTask Processor
 3477 containing the `<htcp:responseAction>` element in the incoming EPR. The EPR is mapped to SOAP
 3478 header fields as follows: The endpoint reference to be used by the human task for submitting its response
 3479 message to is contained in the `<wsa:ReplyTo>` element. The address of the endpoint is contained in the
 3480 `<wsa:Address>` element. The identifier of the instance of the caller to be encoded as reference
 3481 parameters in the response message is nested in the `<wsa:ReferenceParameters>` element. The
 3482 value of the `<wsa:Action>` element to be set by the human task in its response to the caller is in the
 3483 `<htcp:responseAction>` element nested in the `<wsa:Metadata>` element of the EPR.

```

3484 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
3485   xmlns:wsa="http://www.w3.org/2005/08/addressing"
3486   xmlns:htcp="http://docs.oasis-open.org/ns/bpel4people/ws-
3487 humantask/protocol/200803">
3488
3489   <S:Header>
3490     <wsa:ReplyTo>
3491       <wsa:Address>http://example.com/LoanApproval/loan</wsa:Address>
3492     <wsa:ReferenceParameters>
3493       <xmp:MyInstanceID>42</xmp:MyInstanceID>
3494     </wsa:ReferenceParameters>
3495     <wsa:Metadata>
3496       <htcp:responseAction>
3497         http://example.com/LoanApproval/approvalResponse
3498       </htcp:responseAction>
3499     </wsa:Metadata>
3500   </wsa:ReplyTo>
3501 </S:Header>
3502
3503 <S:Body>...</S:Body>
  
```


3504 </S:Envelope>

3505 The following is an example of a response message corresponding to the request message discussed
3506 above. This response is sent from the WS-HumanTask Processor back to the caller. The <wsa:To>
3507 element contains a copy of the <wsa:Address> element of the original request message. The
3508 <wsa:Action> element is copied from the <http:responseAction> element of the original request
3509 message. The reference parameters are copied as standalone elements (the <xmp:MyInstanceID>
3510 element below) out of the <wsa:ReferenceParameters> element of the request message.

```
3511 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"  
3512   xmlns:wsa="http://www.w3.org/2005/08/addressing">  
3513   <S:Header>  
3514     <wsa:To>  
3515       <wsa:Address>http://example.com/LoanApproval/loan</wsa:Address>  
3516     </wsa:To>  
3517     <wsa:Action>  
3518       http://example.com/LoanApproval/approvalResponse  
3519     </wsa:Action>  
3520     <xmp:MyInstanceID wsa:IsReferenceParameter='true'>  
3521       42  
3522     </xmp:MyInstanceID>  
3523   </S:Header>  
3524   <S:Body>...</S:Body>  
3525 </S:Envelope>
```

3526 The following is an example of a request message sent from the caller to the WS-HumanTask Processor
3527 containing the <http:responseOperation> element and corresponding WSDL metadata in the
3528 incoming EPR. The EPR is mapped to SOAP header fields as follows: The endpoint reference to be used
3529 by the WS-HumanTask Processor for submitting its response message to is contained in the
3530 <wsa:ReplyTo> element. The address of the endpoint is contained in the <wsa:Address> element.
3531 The identifier of the instance of the caller to be encoded as reference parameters in the response
3532 message is nested in the <wsa:ReferenceParameters> element. The WSDL metadata of the
3533 endpoint is contained in the <wsdl:definitions> element. The name of the operation of the endpoint
3534 to be used to send the response message to is contained in the <http:responseOperation>
3535 element. Both elements are nested in the <wsa:Metadata> element of the EPR. These elements
3536 provide the basis to determine the value of the action header field to be set by the WS-HumanTask
3537 Processor in its response to the caller.

```
3538 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"  
3539   xmlns:wsa="http://www.w3.org/2005/08/addressing"  
3540   xmlns:http="http://docs.oasis-open.org/ns/bpel4people/ws-  
3541   humantask/protocol/200803">  
3542   <S:Header>  
3543     <wsa:ReplyTo>  
3544       <wsa:Address>http://example.com/LoanApproval/loan</wsa:Address>  
3545     <wsa:ReferenceParameters>  
3546       <xmp:MyInstanceID>42</xmp:MyInstanceID>  
3547     </wsa:ReferenceParameters>  
3548     <wsa:Metadata>  
3549       <wsdl:definitions  
3550         targetNamespace="http://example.com/loanApproval"  
3551         xmlns:wsdl="..." xmlns:soap="...">  
3552         <wsdl:portType name="LoanApprovalPT">  
3553           <wsdl:operation name="approvalResponse">  
3554             <wsdl:input name="approvalInput" ... />  
3555         </wsdl:portType>  
3556       </wsa:Metadata>  
3557     </wsa:ReferenceParameters>  
3558   </S:Header>  
3559 </S:Envelope>
```

```

3560     </wsdl:operation>
3561     ...
3562 </wsdl:portType>
3563
3564 <wsdl:binding name="LoanApprovalSoap"
3565     type="LoanApprovalPT">
3566     ...
3567 </wsdl:binding>
3568
3569 <wsdl:service name="LoanApprovalService">
3570 <wsdl:port name="LA" binding="LoanApprovalSoap">
3571 <soap:address
3572     location="http://example.com/LoanApproval/loan" />
3573 </wsdl:port>
3574     ...
3575 </wsdl:service>
3576 </wsdl:definitions>
3577
3578 <htcp:responseOperation>
3579     approvalResponse
3580 </htcp:responseOperation>
3581
3582 </wsa:Metadata>
3583 </wsa:ReplyTo>
3584
3585 </S:Header>
3586 <S:Body>...</S:Body>
3587 </S:Envelope>

```

3588 The following is an example of a response message corresponding to the request message before; this
3589 response is sent from the WS-HumanTask Processor back to the caller. The `<wsa:To>` element contains
3590 a copy of the `<wsa:Address>` field of the original request message. The reference parameters are
3591 copied as standalone element (the `<xmp:MyInstanceID>` element below) out of the
3592 `<htcp:ReferenceParameters>` element of the request message. The value of the `<wsa:Action>`
3593 element is composed according to [WS-Addr-WSDL] from the target namespace, port type name, name
3594 of the response operation to be used, and name of the input message of this operation given in the code
3595 snippet above.

```

3596 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
3597     xmlns:wsa="http://www.w3.org/2005/08/addressing"
3598     xmlns:htd="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803">
3599 <S:Header>
3600 <wsa:To>http://example.com/LoanApproval/loan</wsa:To>
3601 <wsa:Action>
3602     http://example.com/loanApproval/...
3603     ...LoanApprovalPT/approvalResponse/ApprovalInput
3604 </wsa:Action>
3605 <xmp:MyInstanceID wsa:IsReferenceParameter='true'>
3606     42
3607 </xmp:MyInstanceID>
3608 </S:Header>
3609 <S:Body>...</S:Body>
3610 </S:Envelope>

```

3611

11 Security Considerations

3612 WS-HumanTask does not mandate the use of any specific mechanism or technology for client
3613 authentication. However, a client **MUST** provide a principal or the principal **MUST** be obtainable by the
3614 WS-HumanTask Processor.

3615 When using task APIs via SOAP bindings, compliance with the WS-I Basic Security Profile 1.0 is
3616 **RECOMMENDED**.

3617 **12 Conformance**

3618

3619 The XML schema pointed to by the RDDL document at the namespace URI, defined by this specification,
3620 are considered to be authoritative and take precedence over the XML schema defined in the appendix of
3621 this document.

3622

3623 There are four conformance targets defined as part of this specification: a WS-HumanTask Definition, a
3624 WS-HumanTask Processor, a WS-HumanTask Parent and a WS-HumanTask Client (see section 2.3). In
3625 order to claim conformance with WS-HumanTask 1.1, the conformance targetes MUST comply with all
3626 normative statements in this specification, notably all MUST statements have to be implemented.

3627

A. Portability and Interoperability Considerations

3628 This section illustrates the portability and interoperability aspects addressed by WS-HumanTask:

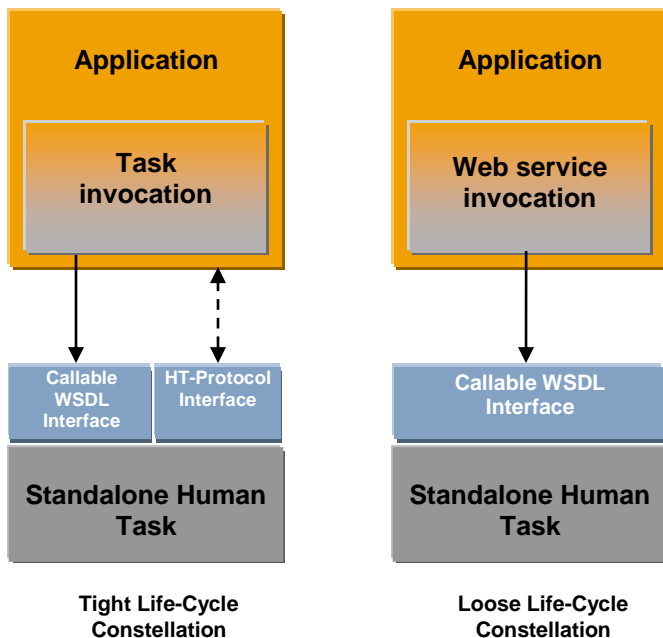
- 3629 • Portability - The ability to take human tasks and notifications created in one vendor's environment
3630 and use them in another vendor's environment.
- 3631 • Interoperability - The capability for multiple components (task infrastructure, task list clients and
3632 applications or processes with human interactions) to interact using well-defined messages and
3633 protocols. This enables combining components from different vendors allowing seamless
3634 execution.

3635 Portability requires support of WS-HumanTask artifacts.

3636 Interoperability between task infrastructure and task list clients is achieved using the operations for client
3637 applications.

3638 Interoperability between applications and task infrastructure from different vendors subsumes two
3639 alternative constellations depending on how tightly the life-cycles of the task and the invoking
3640 application are coupled with each other. This is shown in the figure below:

3641 Tight Life-Cycle Constellation: Applications are human task aware and control the life cycle of tasks.
3642 Interoperability between applications and WS-HumanTask Processors is achieved using the WS-
3643 HumanTask coordination protocol.



3644 Loose Life-Cycle Constellation: Applications use basic Web services protocols to invoke Web services
3645 implemented as human tasks. In this case standard Web services interoperability is achieved and
3646 applications do not control the life cycle of tasks.

3647

B. WS-HumanTask Language Schema

```
3648 <?xml version="1.0" encoding="UTF-8"?>
3649 <!--
3650 Copyright (c) OASIS Open 2009. All Rights Reserved.
3651 -->
3652 <xsd:schema
3653   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3654   xmlns="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803"
3655   targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
3656   humantask/200803"
3657   elementFormDefault="qualified" blockDefault="#all">
3658
3659   <xsd:annotation>
3660     <xsd:documentation>
3661       XML Schema for WS-HumanTask 1.1 - WS-HumanTask Task Definition Language
3662     </xsd:documentation>
3663   </xsd:annotation>
3664
3665   <!-- other namespaces -->
3666   <xsd:import namespace="http://www.w3.org/XML/1998/namespace"
3667     schemaLocation="http://www.w3.org/2001/xml.xsd" />
3668
3669   <!-- base types for extensible elements -->
3670   <xsd:complexType name="tExtensibleElements">
3671     <xsd:sequence>
3672       <xsd:element name="documentation" type="tDocumentation" minOccurs="0"
3673   maxOccurs="unbounded" />
3674       <xsd:any namespace="##other" processContents="lax" minOccurs="0"
3675   maxOccurs="unbounded" />
3676     </xsd:sequence>
3677     <xsd:anyAttribute namespace="##other" processContents="lax" />
3678   </xsd:complexType>
3679
3680   <xsd:complexType name="tDocumentation" mixed="true">
3681     <xsd:sequence>
3682       <xsd:any namespace="##other" processContents="lax" minOccurs="0"
3683   maxOccurs="unbounded" />
3684     </xsd:sequence>
3685     <xsd:attribute ref="xml:lang" />
3686   </xsd:complexType>
3687
3688   <xsd:complexType name="tExtensibleMixedContentElements"
3689     mixed="true">
3690     <xsd:sequence>
3691       <xsd:element name="documentation" type="tDocumentation" minOccurs="0"
3692   maxOccurs="unbounded" />
3693       <xsd:any namespace="##other" processContents="lax" minOccurs="0"
3694   maxOccurs="unbounded" />
3695     </xsd:sequence>
3696     <xsd:anyAttribute namespace="##other" processContents="lax" />
3697   </xsd:complexType>
3698
3699   <!-- human interactions definition -->
3700   <xsd:element name="humanInteractions" type="tHumanInteractions" />
3701   <xsd:complexType name="tHumanInteractions">
```

```

3702     <xsd:complexContent>
3703         <xsd:extension base="tExtensibleElements">
3704             <xsd:sequence>
3705                 <xsd:element name="extensions" type="tExtensions" minOccurs="0" />
3706                 <xsd:element name="import" type="tImport" minOccurs="0"
3707 maxOccurs="unbounded" />
3708                 <xsd:element name="logicalPeopleGroups" type="tLogicalPeopleGroups"
3709 minOccurs="0" />
3710                 <xsd:element name="tasks" type="tTasks" minOccurs="0" />
3711                 <xsd:element name="notifications" type="tNotifications"
3712 minOccurs="0" />
3713             </xsd:sequence>
3714             <xsd:attribute name="targetNamespace" type="xsd:anyURI"
3715 use="required" />
3716             <xsd:attribute name="queryLanguage" type="xsd:anyURI" />
3717             <xsd:attribute name="expressionLanguage" type="xsd:anyURI" />
3718         </xsd:extension>
3719     </xsd:complexContent>
3720 </xsd:complexType>
3721
3722 <xsd:complexType name="tExtensions">
3723     <xsd:complexContent>
3724         <xsd:extension base="tExtensibleElements">
3725             <xsd:sequence>
3726                 <xsd:element name="extension" type="tExtension"
3727 maxOccurs="unbounded" />
3728             </xsd:sequence>
3729         </xsd:extension>
3730     </xsd:complexContent>
3731 </xsd:complexType>
3732
3733 <xsd:complexType name="tExtension">
3734     <xsd:complexContent>
3735         <xsd:extension base="tExtensibleElements">
3736             <xsd:attribute name="namespace" type="xsd:anyURI" use="required" />
3737             <xsd:attribute name="mustUnderstand" type="tBoolean" use="required"
3738 />
3739         </xsd:extension>
3740     </xsd:complexContent>
3741 </xsd:complexType>
3742
3743 <xsd:element name="import" type="tImport" />
3744 <xsd:complexType name="tImport">
3745     <xsd:complexContent>
3746         <xsd:extension base="tExtensibleElements">
3747             <xsd:attribute name="namespace" type="xsd:anyURI" use="optional" />
3748             <xsd:attribute name="location" type="xsd:anyURI" use="optional" />
3749             <xsd:attribute name="importType" type="xsd:anyURI" use="required" />
3750         </xsd:extension>
3751     </xsd:complexContent>
3752 </xsd:complexType>
3753
3754 <xsd:element name="logicalPeopleGroups" type="tLogicalPeopleGroups" />
3755 <xsd:complexType name="tLogicalPeopleGroups">
3756     <xsd:complexContent>
3757         <xsd:extension base="tExtensibleElements">
3758             <xsd:sequence>

```

```

3759     <xsd:element name="logicalPeopleGroup" type="tLogicalPeopleGroup"
3760 maxOccurs="unbounded" />
3761     </xsd:sequence>
3762 </xsd:extension>
3763 </xsd:complexContent>
3764 </xsd:complexType>
3765
3766 <xsd:complexType name="tLogicalPeopleGroup">
3767   <xsd:complexContent>
3768     <xsd:extension base="tExtensibleElements">
3769       <xsd:sequence>
3770         <xsd:element name="parameter" type="tParameter" minOccurs="0"
3771 maxOccurs="unbounded" />
3772       </xsd:sequence>
3773       <xsd:attribute name="name" type="xsd:NCName" use="required" />
3774       <xsd:attribute name="reference" type="xsd:NCName" use="optional" />
3775     </xsd:extension>
3776   </xsd:complexContent>
3777 </xsd:complexType>
3778
3779 <!-- generic human roles used in tasks and notifications -->
3780 <xsd:element name="genericHumanRole" type="tGenericHumanRoleAssignmentBase"
3781 abstract="true" block=""/>
3782
3783   <xsd:element name="potentialOwners" type="tPotentialOwnerAssignment"
3784 substitutionGroup="genericHumanRole"/>
3785   <xsd:element name="excludedOwners" type="tGenericHumanRoleAssignment"
3786 substitutionGroup="genericHumanRole"/>
3787   <xsd:element name="taskInitiator" type="tGenericHumanRoleAssignment"
3788 substitutionGroup="genericHumanRole"/>
3789   <xsd:element name="taskStakeholders" type="tGenericHumanRoleAssignment"
3790 substitutionGroup="genericHumanRole"/>
3791   <xsd:element name="businessAdministrators"
3792 type="tGenericHumanRoleAssignment" substitutionGroup="genericHumanRole"/>
3793   <xsd:element name="recipients" type="tGenericHumanRoleAssignment"
3794 substitutionGroup="genericHumanRole"/>
3795
3796 <xsd:complexType name="tGenericHumanRoleAssignmentBase" block="">
3797   <xsd:complexContent>
3798     <xsd:extension base="tExtensibleElements"/>
3799   </xsd:complexContent>
3800 </xsd:complexType>
3801
3802 <xsd:complexType name="tGenericHumanRoleAssignment">
3803   <xsd:complexContent>
3804     <xsd:extension base="tGenericHumanRoleAssignmentBase">
3805       <xsd:sequence>
3806         <xsd:element name="from" type="tFrom" />
3807       </xsd:sequence>
3808     </xsd:extension>
3809   </xsd:complexContent>
3810 </xsd:complexType>
3811
3812 <xsd:complexType name="tPotentialOwnerAssignment">
3813   <xsd:complexContent>
3814     <xsd:extension base="tGenericHumanRoleAssignmentBase">
3815       <xsd:choice>
3816         <xsd:element name="from" type="tFrom" />

```

```

3817     <xsd:element name="parallel" type="tParallel" />
3818     <xsd:element name="sequence" type="tSequence" />
3819   </xsd:choice>
3820 </xsd:extension>
3821 </xsd:complexContent>
3822 </xsd:complexType>
3823
3824 <!-- routing patterns -->
3825 <xsd:complexType name="tParallel">
3826   <xsd:complexContent>
3827     <xsd:extension base="tExtensibleElements">
3828       <xsd:sequence>
3829         <xsd:element name="completionBehavior" type="tCompletionBehavior"
3830 minOccurs="0" />
3831         <xsd:element name="from" type="tFrom" minOccurs="0"
3832 maxOccurs="unbounded" />
3833         <xsd:choice minOccurs="0" maxOccurs="unbounded">
3834           <xsd:element name="parallel" type="tParallel" />
3835           <xsd:element name="sequence" type="tSequence" />
3836         </xsd:choice>
3837       </xsd:sequence>
3838       <xsd:attribute name="type" type="tRoutingPatternType" />
3839     </xsd:extension>
3840   </xsd:complexContent>
3841 </xsd:complexType>
3842
3843 <xsd:complexType name="tSequence">
3844   <xsd:complexContent>
3845     <xsd:extension base="tExtensibleElements">
3846       <xsd:sequence>
3847         <xsd:element name="completionBehavior" type="tCompletionBehavior"
3848 />
3849         <xsd:element name="from" type="tFrom" minOccurs="0"
3850 maxOccurs="unbounded" />
3851         <xsd:choice minOccurs="0" maxOccurs="unbounded">
3852           <xsd:element name="parallel" type="tParallel" />
3853           <xsd:element name="sequence" type="tSequence" />
3854         </xsd:choice>
3855       </xsd:sequence>
3856       <xsd:attribute name="type" type="tRoutingPatternType" />
3857     </xsd:extension>
3858   </xsd:complexContent>
3859 </xsd:complexType>
3860
3861 <xsd:simpleType name="tRoutingPatternType">
3862   <xsd:restriction base="xsd:string">
3863     <xsd:enumeration value="all" />
3864     <xsd:enumeration value="single" />
3865   </xsd:restriction>
3866 </xsd:simpleType>
3867
3868 <!-- completion behavior -->
3869 <xsd:complexType name="tCompletionBehavior">
3870   <xsd:complexContent>
3871     <xsd:extension base="tExtensibleElements">
3872       <xsd:sequence>
3873         <xsd:element name="completion" type="tCompletion" minOccurs="0"
3874 maxOccurs="unbounded" />

```

```

3875         <xsd:element name="defaultCompletion" type="tDefaultCompletion"
3876 minOccurs="0" />
3877     </xsd:sequence>
3878     <xsd:attribute name="completionAction" type="tPattern" use="optional"
3879 default="automatic" />
3880 </xsd:extension>
3881 </xsd:complexContent>
3882 </xsd:complexType>
3883
3884 <xsd:complexType name="tCompletion">
3885 <xsd:complexContent>
3886 <xsd:extension base="tExtensibleElements">
3887 <xsd:sequence>
3888 <xsd:element name="condition" type="tBoolean-expr" />
3889 <xsd:element name="result" type="tResult" minOccurs="0" />
3890 </xsd:sequence>
3891 </xsd:extension>
3892 </xsd:complexContent>
3893 </xsd:complexType>
3894
3895 <xsd:complexType name="tDefaultCompletion">
3896 <xsd:complexContent>
3897 <xsd:extension base="tExtensibleElements">
3898 <xsd:sequence>
3899 <xsd:element name="result" type="tResult" />
3900 </xsd:sequence>
3901 </xsd:extension>
3902 </xsd:complexContent>
3903 </xsd:complexType>
3904
3905 <!-- result construction -->
3906 <xsd:complexType name="tResult">
3907 <xsd:complexContent>
3908 <xsd:extension base="tExtensibleElements">
3909 <xsd:choice maxOccurs="unbounded">
3910 <xsd:element name="aggregate" type="tAggregate" />
3911 <xsd:element name="copy" type="tCopy" />
3912 </xsd:choice>
3913 </xsd:extension>
3914 </xsd:complexContent>
3915 </xsd:complexType>
3916
3917 <xsd:complexType name="tAggregate">
3918 <xsd:complexContent>
3919 <xsd:extension base="tExtensibleElements">
3920 <xsd:attribute name="part" type="xsd:NCName" use="optional" />
3921 <xsd:attribute name="location" type="xsd:string" use="optional" />
3922 <xsd:attribute name="condition" type="xsd:string" />
3923 <xsd:attribute name="function" type="xsd:string" use="required" />
3924 </xsd:extension>
3925 </xsd:complexContent>
3926 </xsd:complexType>
3927
3928 <xsd:complexType name="tCopy">
3929 <xsd:complexContent>
3930 <xsd:extension base="tExtensibleElements">
3931 <xsd:sequence>
3932 <xsd:element name="from" type="tExpression" />

```



```

3933     <xsd:element name="to" type="tQuery" />
3934   </xsd:sequence>
3935 </xsd:extension>
3936 </xsd:complexContent>
3937 </xsd:complexType>
3938
3939 <!-- human tasks -->
3940 <xsd:element name="tasks" type="tTasks" />
3941 <xsd:complexType name="tTasks">
3942   <xsd:complexContent>
3943     <xsd:extension base="tExtensibleElements">
3944       <xsd:sequence>
3945         <xsd:element name="task" type="tTask" maxOccurs="unbounded" />
3946       </xsd:sequence>
3947     </xsd:extension>
3948   </xsd:complexContent>
3949 </xsd:complexType>
3950
3951 <xsd:complexType name="tTaskBase" abstract="true">
3952   <xsd:complexContent>
3953     <xsd:extension base="tExtensibleElements">
3954       <xsd:sequence>
3955         <xsd:element name="interface" type="tTaskInterface" minOccurs="0"
3956 />
3957         <xsd:element name="messageSchema" type="tMessageSchema"
3958 minOccurs="0" />
3959         <xsd:element name="priority" type="tPriority-expr" minOccurs="0" />
3960         <xsd:element name="peopleAssignments" type="tPeopleAssignments"
3961 minOccurs="0" />
3962         <xsd:element name="completionBehavior" type="tCompletionBehavior"
3963 minOccurs="0" />
3964         <xsd:element name="delegation" type="tDelegation" minOccurs="0" />
3965         <xsd:element name="presentationElements"
3966 type="tPresentationElements" minOccurs="0" />
3967         <xsd:element name="outcome" type="tQuery" minOccurs="0" />
3968         <xsd:element name="searchBy" type="tExpression" minOccurs="0" />
3969         <xsd:element name="renderings" type="tRenderings" minOccurs="0" />
3970         <xsd:element name="deadlines" type="tDeadlines" minOccurs="0" />
3971         <xsd:element name="composition" type="tComposition" minOccurs="0"
3972 />
3973       </xsd:sequence>
3974       <xsd:attribute name="name" type="xsd:NCName" use="required" />
3975       <xsd:attribute name="actualOwnerRequired" type="tBoolean"
3976 use="optional" default="yes" />
3977     </xsd:extension>
3978   </xsd:complexContent>
3979 </xsd:complexType>
3980
3981 <xsd:element name="task" type="tTask" />
3982 <xsd:complexType name="tTask">
3983   <xsd:complexContent>
3984     <xsd:restriction base="tTaskBase">
3985       <xsd:sequence>
3986         <xsd:element name="documentation" type="tDocumentation"
3987 minOccurs="0" maxOccurs="unbounded" />
3988         <xsd:any namespace="##other" processContents="lax" minOccurs="0"
3989 maxOccurs="unbounded" />
3990         <xsd:element name="interface" type="tTaskInterface" />

```

```

3991     <xsd:element name="messageSchema" type="tMessageSchema"
3992 minOccurs="0" maxOccurs="0" />
3993     <xsd:element name="priority" type="tPriority-expr" minOccurs="0" />
3994     <xsd:element name="peopleAssignments" type="tPeopleAssignments"
3995 minOccurs="0" />
3996     <xsd:element name="completionBehavior" type="tCompletionBehavior"
3997 minOccurs="0" />
3998     <xsd:element name="delegation" type="tDelegation" minOccurs="0" />
3999     <xsd:element name="presentationElements"
4000 type="tPresentationElements" minOccurs="0" />
4001     <xsd:element name="outcome" type="tQuery" minOccurs="0" />
4002     <xsd:element name="searchBy" type="tExpression" minOccurs="0" />
4003     <xsd:element name="renderings" type="tRenderings" minOccurs="0" />
4004     <xsd:element name="deadlines" type="tDeadlines" minOccurs="0" />
4005     <xsd:element name="composition" type="tComposition" minOccurs="0"
4006 />
4007     </xsd:sequence>
4008     <xsd:attribute name="name" type="xsd:NCName" use="required" />
4009     <xsd:attribute name="actualOwnerRequired" type="tBoolean"
4010 use="optional" default="yes" />
4011     <xsd:anyAttribute namespace="##other" processContents="lax" />
4012     </xsd:restriction>
4013     </xsd:complexContent>
4014     </xsd:complexType>
4015
4016     <xsd:complexType name="tTaskInterface">
4017     <xsd:complexContent>
4018     <xsd:extension base="tExtensibleElements">
4019     <xsd:attribute name="portType" type="xsd:QName" use="required" />
4020     <xsd:attribute name="operation" type="xsd:NCName" use="required" />
4021     <xsd:attribute name="responsePortType" type="xsd:QName"
4022 use="optional" />
4023     <xsd:attribute name="responseOperation" type="xsd:NCName"
4024 use="optional" />
4025     </xsd:extension>
4026     </xsd:complexContent>
4027     </xsd:complexType>
4028
4029     <!-- presentation elements -->
4030     <xsd:complexType name="tPresentationElements">
4031     <xsd:complexContent>
4032     <xsd:extension base="tExtensibleElements">
4033     <xsd:sequence>
4034     <xsd:element name="name" type="tText" minOccurs="0"
4035 maxOccurs="unbounded" />
4036     <xsd:element name="presentationParameters"
4037 type="tPresentationParameters" minOccurs="0" />
4038     <xsd:element name="subject" type="tText" minOccurs="0"
4039 maxOccurs="unbounded" />
4040     <xsd:element name="description" type="tDescription" minOccurs="0"
4041 maxOccurs="unbounded" />
4042     </xsd:sequence>
4043     </xsd:extension>
4044     </xsd:complexContent>
4045     </xsd:complexType>
4046
4047     <xsd:complexType name="tPresentationParameters">
4048     <xsd:complexContent>

```

```

4049     <xsd:extension base="tExtensibleElements">
4050         <xsd:sequence>
4051             <xsd:element name="presentationParameter"
4052 type="tPresentationParameter" maxOccurs="unbounded" />
4053         </xsd:sequence>
4054         <xsd:attribute name="expressionLanguage" type="xsd:anyURI" />
4055     </xsd:extension>
4056 </xsd:complexContent>
4057 </xsd:complexType>
4058
4059 <xsd:complexType name="tPresentationParameter">
4060     <xsd:complexContent>
4061         <xsd:extension base="tParameter" />
4062     </xsd:complexContent>
4063 </xsd:complexType>
4064
4065 <!-- elements for rendering tasks -->
4066 <xsd:complexType name="tRenderings">
4067     <xsd:complexContent>
4068         <xsd:extension base="tExtensibleElements">
4069             <xsd:sequence>
4070                 <xsd:element name="rendering" type="tRendering"
4071 maxOccurs="unbounded" />
4072             </xsd:sequence>
4073         </xsd:extension>
4074     </xsd:complexContent>
4075 </xsd:complexType>
4076
4077 <xsd:complexType name="tRendering">
4078     <xsd:complexContent>
4079         <xsd:extension base="tExtensibleElements">
4080             <xsd:attribute name="type" type="xsd:QName" use="required" />
4081         </xsd:extension>
4082     </xsd:complexContent>
4083 </xsd:complexType>
4084
4085 <!-- elements for people assignment -->
4086 <xsd:element name="peopleAssignments" type="tPeopleAssignments" />
4087 <xsd:complexType name="tPeopleAssignments">
4088     <xsd:complexContent>
4089         <xsd:extension base="tExtensibleElements">
4090             <xsd:sequence>
4091                 <xsd:element ref="genericHumanRole" minOccurs="0"
4092 maxOccurs="unbounded" />
4093             </xsd:sequence>
4094         </xsd:extension>
4095     </xsd:complexContent>
4096 </xsd:complexType>
4097
4098 <!-- elements for handling timeouts and escalation -->
4099 <xsd:complexType name="tDeadlines">
4100     <xsd:complexContent>
4101         <xsd:extension base="tExtensibleElements">
4102             <xsd:sequence>
4103                 <xsd:element name="startDeadline" type="tDeadline" minOccurs="0"
4104 maxOccurs="unbounded" />
4105                 <xsd:element name="completionDeadline" type="tDeadline"
4106 minOccurs="0" maxOccurs="unbounded" />

```

```

4107     </xsd:sequence>
4108     </xsd:extension>
4109     </xsd:complexContent>
4110 </xsd:complexType>
4111
4112 <xsd:complexType name="tDeadline">
4113     <xsd:complexContent>
4114         <xsd:extension base="tExtensibleElements">
4115             <xsd:sequence>
4116                 <xsd:choice>
4117                     <xsd:element name="for" type="tDuration-expr" />
4118                     <xsd:element name="until" type="tDeadline-expr" />
4119                 </xsd:choice>
4120                 <xsd:element name="escalation" type="tEscalation" minOccurs="0"
4121 maxOccurs="unbounded" />
4122             </xsd:sequence>
4123             <xsd:attribute name="name" type="xsd:NCName" use="required"/>
4124         </xsd:extension>
4125     </xsd:complexContent>
4126 </xsd:complexType>
4127
4128 <xsd:complexType name="tEscalation">
4129     <xsd:complexContent>
4130         <xsd:extension base="tExtensibleElements">
4131             <xsd:sequence>
4132                 <xsd:element name="condition" type="tBoolean-expr" minOccurs="0" />
4133                 <xsd:element name="toParts" type="tToParts" minOccurs="0" />
4134             <xsd:choice>
4135                 <xsd:element name="notification" type="tNotification" />
4136                 <xsd:element name="localNotification" type="tLocalNotification"
4137 />
4138                 <xsd:element name="reassignment" type="tReassignment" />
4139             </xsd:choice>
4140         </xsd:sequence>
4141         <xsd:attribute name="name" type="xsd:NCName" use="required" />
4142     </xsd:extension>
4143 </xsd:complexContent>
4144 </xsd:complexType>
4145
4146 <xsd:complexType name="tLocalNotification">
4147     <xsd:complexContent>
4148         <xsd:extension base="tExtensibleElements">
4149             <xsd:choice>
4150                 <xsd:sequence>
4151                     <xsd:element name="priority" type="tPriority-expr" minOccurs="0"
4152 />
4153                     <xsd:element name="peopleAssignments" type="tPeopleAssignments"
4154 minOccurs="0" />
4155                 </xsd:sequence>
4156             </xsd:choice>
4157             <xsd:attribute name="reference" type="xsd:QName" use="required" />
4158         </xsd:extension>
4159     </xsd:complexContent>
4160 </xsd:complexType>
4161
4162 <xsd:complexType name="tReassignment">
4163     <xsd:complexContent>
4164         <xsd:extension base="tExtensibleElements">

```

```

4165     <xsd:sequence>
4166         <xsd:element ref="potentialOwners" />
4167     </xsd:sequence>
4168 </xsd:extension>
4169 </xsd:complexContent>
4170 </xsd:complexType>
4171
4172 <xsd:complexType name="tToParts">
4173     <xsd:complexContent>
4174         <xsd:extension base="tExtensibleElements">
4175             <xsd:sequence>
4176                 <xsd:element name="toPart" type="tToPart" maxOccurs="unbounded" />
4177             </xsd:sequence>
4178         </xsd:extension>
4179     </xsd:complexContent>
4180 </xsd:complexType>
4181
4182 <xsd:complexType name="tToPart" mixed="true">
4183     <xsd:complexContent>
4184         <xsd:extension base="tExtensibleMixedContentElements">
4185             <xsd:attribute name="name" type="xsd:NCName" use="required" />
4186             <xsd:attribute name="expressionLanguage" type="xsd:anyURI" />
4187         </xsd:extension>
4188     </xsd:complexContent>
4189 </xsd:complexType>
4190
4191 <!-- task delegation -->
4192 <xsd:complexType name="tDelegation">
4193     <xsd:complexContent>
4194         <xsd:extension base="tExtensibleElements">
4195             <xsd:sequence>
4196                 <xsd:element name="from" type="tFrom" minOccurs="0" />
4197             </xsd:sequence>
4198             <xsd:attribute name="potentialDelegates" type="tPotentialDelegates"
4199 use="required" />
4200         </xsd:extension>
4201     </xsd:complexContent>
4202 </xsd:complexType>
4203
4204 <xsd:simpleType name="tPotentialDelegates">
4205     <xsd:restriction base="xsd:string">
4206         <xsd:enumeration value="anybody" />
4207         <xsd:enumeration value="nobody" />
4208         <xsd:enumeration value="potentialOwners" />
4209         <xsd:enumeration value="other" />
4210     </xsd:restriction>
4211 </xsd:simpleType>
4212
4213 <!-- composite tasks -->
4214 <xsd:complexType name="tComposition">
4215     <xsd:complexContent>
4216         <xsd:extension base="tExtensibleElements">
4217             <xsd:sequence>
4218                 <xsd:element name="subtask" type="tSubtask" maxOccurs="unbounded"
4219 />
4220             </xsd:sequence>
4221             <xsd:attribute name="type" type="tCompositionType" use="optional"
4222 default="sequential" />

```

```

4223     <xsd:attribute name="instantiationPattern" type="tPattern"
4224 use="optional" default="manual" />
4225     </xsd:extension>
4226   </xsd:complexContent>
4227 </xsd:complexType>
4228
4229 <xsd:simpleType name="tCompositionType">
4230   <xsd:restriction base="xsd:string">
4231     <xsd:enumeration value="sequential" />
4232     <xsd:enumeration value="parallel" />
4233   </xsd:restriction>
4234 </xsd:simpleType>
4235
4236 <xsd:simpleType name="tPattern">
4237   <xsd:restriction base="xsd:string">
4238     <xsd:enumeration value="manual" />
4239     <xsd:enumeration value="automatic" />
4240   </xsd:restriction>
4241 </xsd:simpleType>
4242
4243 <xsd:complexType name="tSubtask">
4244   <xsd:complexContent>
4245     <xsd:extension base="tExtensibleElements">
4246       <xsd:choice>
4247         <xsd:element name="task" type="tTask"/>
4248         <xsd:element name="localTask" type="tLocalTask" />
4249       </xsd:choice>
4250       <xsd:attribute name="name" type="xsd:NCName" use="required" />
4251     </xsd:extension>
4252   </xsd:complexContent>
4253 </xsd:complexType>
4254
4255 <xsd:complexType name="tLocalTask">
4256   <xsd:complexContent>
4257     <xsd:extension base="tExtensibleElements">
4258       <xsd:sequence>
4259         <xsd:element name="priority" type="tPriority-expr" minOccurs="0" />
4260         <xsd:element name="peopleAssignments" type="tPeopleAssignments"
4261 minOccurs="0" />
4262       </xsd:sequence>
4263       <xsd:attribute name="reference" type="xsd:QName" use="required" />
4264     </xsd:extension>
4265   </xsd:complexContent>
4266 </xsd:complexType>
4267
4268 <!-- lean tasks -->
4269 <xsd:element name="leanTask" type="tLeanTask"/>
4270 <xsd:complexType name="tLeanTask">
4271   <xsd:complexContent>
4272     <xsd:restriction base="tTaskBase">
4273       <xsd:sequence>
4274         <xsd:element name="documentation" type="tDocumentation"
4275 minOccurs="0" maxOccurs="unbounded" />
4276         <xsd:any namespace="##other" processContents="lax" minOccurs="0"
4277 maxOccurs="unbounded" />
4278         <xsd:element name="interface" type="tTaskInterface" minOccurs="0"
4279 maxOccurs="0" />
4280         <xsd:element name="messageSchema" type="tMessageSchema" />

```

```

4281         <xsd:element name="priority" type="tPriority-expr" minOccurs="0" />
4282         <xsd:element name="peopleAssignments" type="tPeopleAssignments"
4283 minOccurs="0" />
4284         <xsd:element name="delegation" type="tDelegation" minOccurs="0" />
4285         <xsd:element name="presentationElements"
4286 type="tPresentationElements" minOccurs="0" />
4287         <xsd:element name="outcome" type="tQuery" minOccurs="0" />
4288         <xsd:element name="searchBy" type="tExpression" minOccurs="0" />
4289         <xsd:element name="renderings" type="tRenderings" minOccurs="0" />
4290         <xsd:element name="deadlines" type="tDeadlines" minOccurs="0" />
4291         <xsd:element name="composition" type="tComposition" minOccurs="0"
4292 maxOccurs="0" />
4293     </xsd:sequence>
4294     <xsd:attribute name="name" type="xsd:NCName" use="required" />
4295     <xsd:attribute name="actualOwnerRequired" type="tBoolean"
4296 use="optional" default="yes" />
4297     <xsd:anyAttribute namespace="##other" processContents="lax" />
4298 </xsd:restriction>
4299 </xsd:complexContent>
4300 </xsd:complexType>
4301
4302 <xsd:complexType name="tMessageSchema">
4303     <xsd:complexContent>
4304         <xsd:extension base="tExtensibleElements">
4305             <xsd:sequence>
4306                 <xsd:element name="messageField" type="tMessageField"
4307 minOccurs="0" maxOccurs="unbounded" />
4308             </xsd:sequence>
4309         </xsd:extension>
4310     </xsd:complexContent>
4311 </xsd:complexType>
4312
4313 <xsd:complexType name="tMessageField">
4314     <xsd:complexContent>
4315         <xsd:extension base="tExtensibleElements">
4316             <xsd:sequence>
4317                 <xsd:element name="messageDisplay" type="tMessageDisplay"
4318 maxOccurs="unbounded" />
4319                 <xsd:element name="messageChoice" type="tMessageChoice"
4320 minOccurs="0" maxOccurs="unbounded" />
4321             </xsd:sequence>
4322             <xsd:attribute name="name" type="xsd:NCName" />
4323             <xsd:attribute name="type" type="xsd:QName" />
4324         </xsd:extension>
4325     </xsd:complexContent>
4326 </xsd:complexType>
4327
4328 <xsd:complexType name="tMessageChoice">
4329     <xsd:complexContent>
4330         <xsd:extension base="tExtensibleElements">
4331             <xsd:sequence>
4332                 <xsd:element name="messageDisplay" type="tMessageDisplay"
4333 maxOccurs="unbounded" />
4334             </xsd:sequence>
4335             <xsd:attribute name="value" type="xsd:anySimpleType" />
4336         </xsd:extension>
4337     </xsd:complexContent>
4338 </xsd:complexType>

```



```

4339
4340 <xsd:complexType name="tMessageDisplay" mixed="true">
4341   <xsd:complexContent>
4342     <xsd:extension base="tExtensibleMixedContentElements">
4343       <xsd:attribute ref="xml:lang" />
4344     </xsd:extension>
4345   </xsd:complexContent>
4346 </xsd:complexType>
4347
4348 <!-- notifications -->
4349 <xsd:element name="notifications" type="tNotifications" />
4350 <xsd:complexType name="tNotifications">
4351   <xsd:complexContent>
4352     <xsd:extension base="tExtensibleElements">
4353       <xsd:sequence>
4354         <xsd:element name="notification" type="tNotification"
4355 maxOccurs="unbounded" />
4356       </xsd:sequence>
4357     </xsd:extension>
4358   </xsd:complexContent>
4359 </xsd:complexType>
4360
4361 <xsd:element name="notification" type="tNotification" />
4362 <xsd:complexType name="tNotification">
4363   <xsd:complexContent>
4364     <xsd:extension base="tExtensibleElements">
4365       <xsd:sequence>
4366         <xsd:element name="interface" type="tNotificationInterface" />
4367         <xsd:element name="priority" type="tPriority-expr" minOccurs="0" />
4368         <xsd:element name="peopleAssignments" type="tPeopleAssignments" />
4369         <xsd:element name="presentationElements"
4370 type="tPresentationElements" />
4371         <xsd:element name="renderings" type="tRenderings" minOccurs="0" />
4372       </xsd:sequence>
4373       <xsd:attribute name="name" type="xsd:NCName" use="required" />
4374     </xsd:extension>
4375   </xsd:complexContent>
4376 </xsd:complexType>
4377
4378 <xsd:complexType name="tNotificationInterface">
4379   <xsd:complexContent>
4380     <xsd:extension base="tExtensibleElements">
4381       <xsd:attribute name="portType" type="xsd:QName" use="required" />
4382       <xsd:attribute name="operation" type="xsd:NCName" use="required" />
4383     </xsd:extension>
4384   </xsd:complexContent>
4385 </xsd:complexType>
4386
4387 <!-- miscellaneous helper types -->
4388 <xsd:complexType name="tText" mixed="true">
4389   <xsd:complexContent>
4390     <xsd:extension base="tExtensibleMixedContentElements">
4391       <xsd:attribute ref="xml:lang" />
4392     </xsd:extension>
4393   </xsd:complexContent>
4394 </xsd:complexType>
4395
4396 <xsd:complexType name="tDescription" mixed="true">

```



```

4397     <xsd:complexContent>
4398         <xsd:extension base="tExtensibleMixedContentElements">
4399             <xsd:attribute ref="xml:lang" />
4400             <xsd:attribute name="contentType" type="xsd:string" />
4401         </xsd:extension>
4402     </xsd:complexContent>
4403 </xsd:complexType>
4404
4405 <xsd:complexType name="tFrom" mixed="true">
4406     <xsd:complexContent>
4407         <xsd:extension base="tExtensibleMixedContentElements">
4408             <xsd:sequence>
4409                 <xsd:choice>
4410                     <xsd:element name="argument" type="tArgument" minOccurs="0"
4411 maxOccurs="unbounded"/>
4412                     <xsd:element name="literal" type="tLiteral" minOccurs="0" />
4413                 </xsd:choice>
4414             </xsd:sequence>
4415             <xsd:attribute name="expressionLanguage" type="xsd:anyURI" />
4416             <xsd:attribute name="logicalPeopleGroup" type="xsd:NCName" />
4417         </xsd:extension>
4418     </xsd:complexContent>
4419 </xsd:complexType>
4420
4421 <xsd:complexType name="tArgument">
4422     <xsd:complexContent>
4423         <xsd:extension base="tExtensibleMixedContentElements">
4424             <xsd:attribute name="name" type="xsd:NCName" />
4425             <xsd:attribute name="expressionLanguage" type="xsd:anyURI" />
4426         </xsd:extension>
4427     </xsd:complexContent>
4428 </xsd:complexType>
4429
4430 <xsd:complexType name="tParameter" mixed="true">
4431     <xsd:complexContent>
4432         <xsd:extension base="tExtensibleMixedContentElements">
4433             <xsd:attribute name="name" type="xsd:NCName" use="required" />
4434             <xsd:attribute name="type" type="xsd:QName" use="required" />
4435         </xsd:extension>
4436     </xsd:complexContent>
4437 </xsd:complexType>
4438
4439 <xsd:complexType name="tLiteral" mixed="true">
4440     <xsd:sequence>
4441         <xsd:any namespace="##any" processContents="lax"/>
4442     </xsd:sequence>
4443     <xsd:anyAttribute namespace="##other" processContents="lax" />
4444 </xsd:complexType>
4445
4446 <xsd:complexType name="tQuery" mixed="true">
4447     <xsd:complexContent>
4448         <xsd:extension base="tExtensibleMixedContentElements">
4449             <xsd:attribute name="part" />
4450             <xsd:attribute name="queryLanguage" type="xsd:anyURI" />
4451         </xsd:extension>
4452     </xsd:complexContent>
4453 </xsd:complexType>
4454

```

```

4455 <xsd:complexType name="tExpression" mixed="true">
4456   <xsd:complexContent>
4457     <xsd:extension base="tExtensibleMixedContentElements">
4458       <xsd:attribute name="expressionLanguage" type="xsd:anyURI" />
4459     </xsd:extension>
4460   </xsd:complexContent>
4461 </xsd:complexType>
4462
4463 <xsd:element name="priority" type="tPriority-expr" />
4464 <xsd:complexType name="tPriority-expr" mixed="true">
4465   <xsd:complexContent mixed="true">
4466     <xsd:extension base="tExpression" />
4467   </xsd:complexContent>
4468 </xsd:complexType>
4469
4470 <xsd:complexType name="tBoolean-expr" mixed="true">
4471   <xsd:complexContent mixed="true">
4472     <xsd:extension base="tExpression" />
4473   </xsd:complexContent>
4474 </xsd:complexType>
4475
4476 <xsd:complexType name="tDuration-expr" mixed="true">
4477   <xsd:complexContent mixed="true">
4478     <xsd:extension base="tExpression" />
4479   </xsd:complexContent>
4480 </xsd:complexType>
4481
4482 <xsd:complexType name="tDeadline-expr" mixed="true">
4483   <xsd:complexContent mixed="true">
4484     <xsd:extension base="tExpression" />
4485   </xsd:complexContent>
4486 </xsd:complexType>
4487
4488 <xsd:simpleType name="tBoolean">
4489   <xsd:restriction base="xsd:string">
4490     <xsd:enumeration value="yes" />
4491     <xsd:enumeration value="no" />
4492   </xsd:restriction>
4493 </xsd:simpleType>
4494
4495 </xsd:schema>

```

C. WS-HumanTask Data Types Schema

```

4497 <?xml version="1.0" encoding="UTF-8"?>
4498 <!--
4499 Copyright (c) OASIS Open 2009. All Rights Reserved.
4500 -->
4501 <xsd:schema
4502   targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
4503 humantask/types/200803"
4504   xmlns="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/types/200803"
4505   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
4506   elementFormDefault="qualified"
4507   blockDefault="#all">
4508
4509   <xsd:annotation>
4510     <xsd:documentation>
4511       XML Schema for WS-HumanTask 1.1 - WS-HumanTask Data Type Definitions
4512     </xsd:documentation>
4513   </xsd:annotation>
4514
4515   <!-- other namespaces -->
4516   <xsd:import namespace="http://www.w3.org/XML/1998/namespace"
4517 schemaLocation="http://www.w3.org/2001/xml.xsd"/>
4518
4519   <!-- data types for attachment operations -->
4520   <xsd:element name="attachmentInfo" type="tAttachmentInfo"/>
4521   <xsd:complexType name="tAttachmentInfo">
4522     <xsd:sequence>
4523       <xsd:element name="identifier" type="xsd:anyURI"/>
4524       <xsd:element name="name" type="xsd:string"/>
4525       <xsd:element name="accessType" type="xsd:string"/>
4526       <xsd:element name="contentType" type="xsd:string"/>
4527       <xsd:element name="contentCategory" type="xsd:anyURI"/>
4528       <xsd:element name="attachedTime" type="xsd:dateTime"/>
4529       <xsd:element name="attachedBy" type="tUser"/>
4530       <xsd:any namespace="##other" processContents="lax" minOccurs="0"
4531 maxOccurs="unbounded"/>
4532     </xsd:sequence>
4533   </xsd:complexType>
4534   <xsd:element name="attachment" type="tAttachment"/>
4535   <xsd:complexType name="tAttachment">
4536     <xsd:sequence>
4537       <xsd:element ref="attachmentInfo"/>
4538       <xsd:element name="value" type="xsd:anyType"/>
4539     </xsd:sequence>
4540   </xsd:complexType>
4541
4542   <!-- data types for comments -->
4543   <xsd:element name="comment" type="tComment"/>
4544   <xsd:complexType name="tComment">
4545     <xsd:sequence>
4546       <xsd:element name="id" type="xsd:anyURI"/>
4547       <xsd:element name="addedTime" type="xsd:dateTime"/>
4548       <xsd:element name="addedBy" type="tUser"/>
4549       <xsd:element name="lastModifiedTime" type="xsd:dateTime"/>

```

```

4550     <xsd:element name="lastModifiedBy" type="tUser"/>
4551     <xsd:element name="text" type="xsd:string"/>
4552     <xsd:any namespace="##other" processContents="lax" minOccurs="0"
4553 maxOccurs="unbounded"/>
4554   </xsd:sequence>
4555 </xsd:complexType>
4556
4557 <!-- data types for simple query operations -->
4558 <xsd:element name="taskAbstract" type="tTaskAbstract"/>
4559 <xsd:complexType name="tTaskAbstract">
4560   <xsd:sequence>
4561     <xsd:element name="id" type="xsd:anyURI"/>
4562     <xsd:element name="taskType" type="xsd:string"/>
4563     <xsd:element name="name" type="xsd:QName"/>
4564     <xsd:element name="status" type="tStatus"/>
4565     <xsd:element name="priority" type="tPriority" minOccurs="0"/>
4566     <xsd:element name="createdTime" type="xsd:dateTime"/>
4567     <xsd:element name="activationTime" type="xsd:dateTime" minOccurs="0"/>
4568     <xsd:element name="expirationTime" type="xsd:dateTime" minOccurs="0"/>
4569     <xsd:element name="isSkipable" type="xsd:boolean" minOccurs="0"/>
4570     <xsd:element name="hasPotentialOwners" type="xsd:boolean"
4571 minOccurs="0"/>
4572     <xsd:element name="startByTimeExists" type="xsd:boolean"
4573 minOccurs="0"/>
4574     <xsd:element name="completeByTimeExists" type="xsd:boolean"
4575 minOccurs="0"/>
4576     <xsd:element name="presentationName" type="tPresentationName"
4577 minOccurs="0"/>
4578     <xsd:element name="presentationSubject" type="tPresentationSubject"
4579 minOccurs="0"/>
4580     <xsd:element name="renderingMethodExists" type="xsd:boolean"/>
4581     <xsd:element name="hasOutput" type="xsd:boolean" minOccurs="0"/>
4582     <xsd:element name="hasFault" type="xsd:boolean" minOccurs="0"/>
4583     <xsd:element name="hasAttachments" type="xsd:boolean" minOccurs="0"/>
4584     <xsd:element name="hasComments" type="xsd:boolean" minOccurs="0"/>
4585     <xsd:element name="escalated" type="xsd:boolean" minOccurs="0"/>
4586     <xsd:element name="outcome" type="xsd:string" minOccurs="0"/>
4587     <xsd:element name="parentTaskId" type="xsd:anyURI" minOccurs="0"/>
4588     <xsd:element name="hasSubTasks" type="xsd:boolean" minOccurs="0"/>
4589     <xsd:any namespace="##other" processContents="lax" minOccurs="0"
4590 maxOccurs="unbounded"/>
4591   </xsd:sequence>
4592 </xsd:complexType>
4593 <xsd:element name="taskDetails" type="tTaskDetails"/>
4594 <xsd:complexType name="tTaskDetails">
4595   <xsd:sequence>
4596     <xsd:element name="id" type="xsd:anyURI"/>
4597     <xsd:element name="taskType" type="xsd:string"/>
4598     <xsd:element name="name" type="xsd:QName"/>
4599     <xsd:element name="status" type="tStatus"/>
4600     <xsd:element name="priority" type="tPriority" minOccurs="0"/>
4601     <xsd:element name="taskInitiator" type="tUser" minOccurs="0"/>
4602     <xsd:element name="taskStakeholders" type="tOrganizationalEntity"
4603 minOccurs="0"/>
4604     <xsd:element name="potentialOwners" type="tOrganizationalEntity"
4605 minOccurs="0"/>
4606     <xsd:element name="businessAdministrators" type="tOrganizationalEntity"
4607 minOccurs="0"/>

```

```

4608     <xsd:element name="actualOwner" type="tUser" minOccurs="0"/>
4609     <xsd:element name="notificationRecipients" type="tOrganizationalEntity"
4610 minOccurs="0"/>
4611     <xsd:element name="createdTime" type="xsd:dateTime"/>
4612     <xsd:element name="createdBy" type="tUser" minOccurs="0"/>
4613     <xsd:element name="lastModifiedTime" type="xsd:dateTime"/>
4614     <xsd:element name="lastModifiedBy" type="tUser" minOccurs="0"/>
4615     <xsd:element name="activationTime" type="xsd:dateTime" minOccurs="0"/>
4616     <xsd:element name="expirationTime" type="xsd:dateTime" minOccurs="0"/>
4617     <xsd:element name="isSkipable" type="xsd:boolean" minOccurs="0"/>
4618     <xsd:element name="hasPotentialOwners" type="xsd:boolean"
4619 minOccurs="0"/>
4620     <xsd:element name="startByTimeExists" type="xsd:boolean"
4621 minOccurs="0"/>
4622     <xsd:element name="completeByTimeExists" type="xsd:boolean"
4623 minOccurs="0"/>
4624     <xsd:element name="presentationName" type="tPresentationName"
4625 minOccurs="0"/>
4626     <xsd:element name="presentationSubject" type="tPresentationSubject"
4627 minOccurs="0"/>
4628     <xsd:element name="renderingMethodExists" type="xsd:boolean"/>
4629     <xsd:element name="hasOutput" type="xsd:boolean" minOccurs="0"/>
4630     <xsd:element name="hasFault" type="xsd:boolean" minOccurs="0"/>
4631     <xsd:element name="hasAttachments" type="xsd:boolean" minOccurs="0"/>
4632     <xsd:element name="hasComments" type="xsd:boolean" minOccurs="0"/>
4633     <xsd:element name="escalated" type="xsd:boolean" minOccurs="0"/>
4634     <xsd:element name="searchBy" type="xsd:string" minOccurs="0"/>
4635     <xsd:element name="outcome" type="xsd:string" minOccurs="0"/>
4636     <xsd:element name="parentTaskId" type="xsd:anyURI" minOccurs="0"/>
4637     <xsd:element name="hasSubTasks" type="xsd:boolean" minOccurs="0"/>
4638     <xsd:any namespace="##other" processContents="lax" minOccurs="0"
4639 maxOccurs="unbounded"/>
4640 </xsd:sequence>
4641 </xsd:complexType>
4642 <xsd:simpleType name="tPresentationName">
4643   <xsd:annotation>
4644     <xsd:documentation>length-restricted string</xsd:documentation>
4645   </xsd:annotation>
4646   <xsd:restriction base="xsd:string">
4647     <xsd:maxLength value="64"/>
4648     <xsd:whiteSpace value="preserve"/>
4649   </xsd:restriction>
4650 </xsd:simpleType>
4651 <xsd:simpleType name="tPresentationSubject">
4652   <xsd:annotation>
4653     <xsd:documentation>length-restricted string</xsd:documentation>
4654   </xsd:annotation>
4655   <xsd:restriction base="xsd:string">
4656     <xsd:maxLength value="254"/>
4657     <xsd:whiteSpace value="preserve"/>
4658   </xsd:restriction>
4659 </xsd:simpleType>
4660 <xsd:simpleType name="tStatus">
4661   <xsd:restriction base="xsd:string"/>
4662 </xsd:simpleType>
4663 <xsd:simpleType name="tPredefinedStatus">
4664   <xsd:annotation>
4665     <xsd:documentation>for documentation only</xsd:documentation>

```

```

4666 </xsd:annotation>
4667 <xsd:restriction base="xsd:string">
4668   <xsd:enumeration value="CREATED"/>
4669   <xsd:enumeration value="READY"/>
4670   <xsd:enumeration value="RESERVED"/>
4671   <xsd:enumeration value="IN_PROGRESS"/>
4672   <xsd:enumeration value="SUSPENDED"/>
4673   <xsd:enumeration value="COMPLETED"/>
4674   <xsd:enumeration value="FAILED"/>
4675   <xsd:enumeration value="ERROR"/>
4676   <xsd:enumeration value="EXITED"/>
4677   <xsd:enumeration value="OBSOLETE"/>
4678 </xsd:restriction>
4679 </xsd:simpleType>
4680 <xsd:simpleType name="tPriority">
4681   <xsd:restriction base="xsd:integer">
4682     <xsd:minInclusive value="0"/>
4683     <xsd:maxInclusive value="10"/>
4684   </xsd:restriction>
4685 </xsd:simpleType>
4686 <xsd:complexType name="tTime">
4687   <xsd:choice>
4688     <xsd:element name="timePeriod" type="xsd:duration"/>
4689     <xsd:element name="pointOfTime" type="xsd:dateTime"/>
4690   </xsd:choice>
4691 </xsd:complexType>
4692
4693 <!-- task operations -->
4694 <xsd:complexType name="tTaskOperations">
4695   <xsd:choice maxOccurs="unbounded">
4696     <xsd:element name="activate" type="tTaskOperation"/>
4697     <xsd:element name="addAttachment" type="tTaskOperation"/>
4698     <xsd:element name="addComment" type="tTaskOperation"/>
4699     <xsd:element name="claim" type="tTaskOperation"/>
4700     <xsd:element name="complete" type="tTaskOperation"/>
4701     <xsd:element name="delegate" type="tTaskOperation"/>
4702     <xsd:element name="deleteAttachment" type="tTaskOperation"/>
4703     <xsd:element name="deleteComment" type="tTaskOperation"/>
4704     <xsd:element name="deleteFault" type="tTaskOperation"/>
4705     <xsd:element name="deleteOutput" type="tTaskOperation"/>
4706     <xsd:element name="fail" type="tTaskOperation"/>
4707     <xsd:element name="forward" type="tTaskOperation"/>
4708     <xsd:element name="getAttachment" type="tTaskOperation"/>
4709     <xsd:element name="getAttachmentInfos" type="tTaskOperation"/>
4710     <xsd:element name="getComments" type="tTaskOperation"/>
4711     <xsd:element name="getFault" type="tTaskOperation"/>
4712     <xsd:element name="getInput" type="tTaskOperation"/>
4713     <xsd:element name="getOutcome" type="tTaskOperation"/>
4714     <xsd:element name="getOutput" type="tTaskOperation"/>
4715     <xsd:element name="getParentTask" type="tTaskOperation"/>
4716     <xsd:element name="getParentTaskIdentifier" type="tTaskOperation"/>
4717     <xsd:element name="getRendering" type="tTaskOperation"/>
4718     <xsd:element name="getRenderingTypes" type="tTaskOperation"/>
4719     <xsd:element name="getSubtaskIdentifiers" type="tTaskOperation"/>
4720     <xsd:element name="getSubtasks" type="tTaskOperation"/>
4721     <xsd:element name="getTaskDescription" type="tTaskOperation"/>
4722     <xsd:element name="getTaskDetails" type="tTaskOperation"/>
4723     <xsd:element name="getTaskHistory" type="tTaskOperation"/>

```



```

4724     <xsd:element name="getTaskInstanceData" type="tTaskOperation"/>
4725     <xsd:element name="hasSubtasks" type="tTaskOperation"/>
4726     <xsd:element name="instantiateSubtask" type="tTaskOperation"/>
4727     <xsd:element name="isSubtask" type="tTaskOperation"/>
4728     <xsd:element name="nominate" type="tTaskOperation"/>
4729     <xsd:element name="release" type="tTaskOperation"/>
4730     <xsd:element name="remove" type="tTaskOperation"/>
4731     <xsd:element name="resume" type="tTaskOperation"/>
4732     <xsd:element name="setFault" type="tTaskOperation"/>
4733     <xsd:element name="setGenericHumanRole" type="tTaskOperation"/>
4734     <xsd:element name="setOutput" type="tTaskOperation"/>
4735     <xsd:element name="setPriority" type="tTaskOperation"/>
4736     <xsd:element name="setTaskCompletionDeadlineExpression"
4737 type="tTaskOperation"/>
4738     <xsd:element name="setTaskCompletionDurationExpression"
4739 type="tTaskOperation"/>
4740     <xsd:element name="setTaskStartDeadlineExpression"
4741 type="tTaskOperation"/>
4742     <xsd:element name="setTaskStartDurationExpression"
4743 type="tTaskOperation"/>
4744     <xsd:element name="skip" type="tTaskOperation"/>
4745     <xsd:element name="start" type="tTaskOperation"/>
4746     <xsd:element name="stop" type="tTaskOperation"/>
4747     <xsd:element name="suspend" type="tTaskOperation"/>
4748     <xsd:element name="suspendUntil" type="tTaskOperation"/>
4749     <xsd:element name="updateComment" type="tTaskOperation"/>
4750     <xsd:any namespace="##other" processContents="lax"/>
4751 </xsd:choice>
4752 </xsd:complexType>
4753 <xsd:complexType name="tTaskOperation">
4754   <xsd:complexContent>
4755     <xsd:restriction base="xsd:anyType"/>
4756   </xsd:complexContent>
4757 </xsd:complexType>
4758
4759 <!-- data types for advanced query operations -->
4760 <xsd:element name="taskQueryResultSet" type="tTaskQueryResultSet"/>
4761 <xsd:complexType name="tTaskQueryResultSet">
4762   <xsd:sequence>
4763     <xsd:element name="row" type="tTaskQueryResultRow" minOccurs="0"
4764 maxOccurs="unbounded"/>
4765   </xsd:sequence>
4766 </xsd:complexType>
4767 <xsd:complexType name="tTaskQueryResultRow">
4768   <xsd:choice minOccurs="0" maxOccurs="unbounded">
4769     <xsd:element name="id" type="xsd:anyURI"/>
4770     <xsd:element name="taskType" type="xsd:string"/>
4771     <xsd:element name="name" type="xsd:QName"/>
4772     <xsd:element name="status" type="tStatus"/>
4773     <xsd:element name="priority" type="tPriority"/>
4774     <xsd:element name="taskInitiator" type="tOrganizationalEntity"/>
4775     <xsd:element name="taskStakeholders" type="tOrganizationalEntity"/>
4776     <xsd:element name="potentialOwners" type="tOrganizationalEntity"/>
4777     <xsd:element name="businessAdministrators"
4778 type="tOrganizationalEntity"/>
4779     <xsd:element name="actualOwner" type="tUser"/>
4780     <xsd:element name="notificationRecipients"
4781 type="tOrganizationalEntity"/>

```

```

4782     <xsd:element name="createdTime" type="xsd:dateTime"/>
4783     <xsd:element name="createdBy" type="tUser"/>
4784     <xsd:element name="lastModifiedTime" type="xsd:dateTime"/>
4785     <xsd:element name="lastModifiedBy" type="tUser"/>
4786     <xsd:element name="activationTime" type="xsd:dateTime"/>
4787     <xsd:element name="expirationTime" type="xsd:dateTime"/>
4788     <xsd:element name="isSkipable" type="xsd:boolean"/>
4789     <xsd:element name="hasPotentialOwners" type="xsd:boolean"/>
4790     <xsd:element name="startByTime" type="xsd:dateTime"/>
4791     <xsd:element name="completeByTime" type="xsd:dateTime"/>
4792     <xsd:element name="presentationName" type="tPresentationName"/>
4793     <xsd:element name="presentationSubject" type="tPresentationSubject"/>
4794     <xsd:element name="renderingMethodName" type="xsd:QName"/>
4795     <xsd:element name="hasOutput" type="xsd:boolean"/>
4796     <xsd:element name="hasFault" type="xsd:boolean"/>
4797     <xsd:element name="hasAttachments" type="xsd:boolean"/>
4798     <xsd:element name="hasComments" type="xsd:boolean"/>
4799     <xsd:element name="escalated" type="xsd:boolean"/>
4800     <xsd:element name="parentTaskId" type="xsd:anyURI"/>
4801     <xsd:element name="hasSubtasks" type="xsd:boolean"/>
4802     <xsd:element name="searchBy" type="xsd:string"/>
4803     <xsd:element name="outcome" type="xsd:string"/>
4804     <xsd:element name="taskOperations" type="tTaskOperations"/>
4805     <xsd:any namespace="##other" processContents="lax"/>
4806   </xsd:choice>
4807 </xsd:complexType>
4808 <xsd:complexType name="tFault">
4809   <xsd:sequence>
4810     <xsd:element name="faultName" type="xsd:NCName"/>
4811     <xsd:element name="faultData" type="xsd:anyType"/>
4812   </xsd:sequence>
4813 </xsd:complexType>
4814
4815 <!-- elements and types for organizational entities -->
4816 <xsd:element name="organizationalEntity" type="tOrganizationalEntity"/>
4817 <xsd:complexType name="tOrganizationalEntity">
4818   <xsd:choice maxOccurs="unbounded">
4819     <xsd:element name="user" type="tUser"/>
4820     <xsd:element name="group" type="tGroup"/>
4821   </xsd:choice>
4822 </xsd:complexType>
4823 <xsd:element name="user" type="tUser"/>
4824 <xsd:simpleType name="tUser">
4825   <xsd:restriction base="xsd:string"/>
4826 </xsd:simpleType>
4827 <xsd:element name="group" type="tGroup"/>
4828 <xsd:simpleType name="tGroup">
4829   <xsd:restriction base="xsd:string"/>
4830 </xsd:simpleType>
4831
4832 <!-- input or output message part data -->
4833 <xsd:element name="part" type="tPart"/>
4834 <xsd:complexType name="tPart" mixed="true">
4835   <xsd:sequence>
4836     <xsd:any processContents="skip" minOccurs="0"/>
4837   </xsd:sequence>
4838   <xsd:attribute name="name" type="xsd:NCName" use="required"/>
4839 </xsd:complexType>

```



```

4840
4841 <!-- type container element for one or more message parts -->
4842 <xsd:complexType name="tMessagePartsData">
4843   <xsd:sequence>
4844     <xsd:element ref="part" minOccurs="0" maxOccurs="unbounded"/>
4845   </xsd:sequence>
4846 </xsd:complexType>
4847 <xsd:complexType name="tFaultData">
4848   <xsd:sequence>
4849     <xsd:element name="faultName" type="xsd:NCName"/>
4850     <xsd:element name="faultData" type="xsd:anyType"/>
4851   </xsd:sequence>
4852 </xsd:complexType>
4853 <xsd:element name="attachmentInfos" type="tAttachmentInfos"/>
4854 <xsd:complexType name="tAttachmentInfos">
4855   <xsd:sequence>
4856     <xsd:element name="info" type="tAttachmentInfo" minOccurs="0"
4857 maxOccurs="unbounded"/>
4858   </xsd:sequence>
4859 </xsd:complexType>
4860 <xsd:element name="comments" type="tComments"/>
4861 <xsd:complexType name="tComments">
4862   <xsd:sequence>
4863     <xsd:element ref="comment" minOccurs="0" maxOccurs="unbounded"/>
4864   </xsd:sequence>
4865 </xsd:complexType>
4866 <xsd:element name="renderingType" type="xsd:QName"/>
4867 <xsd:complexType name="tRenderingTypes">
4868   <xsd:sequence>
4869     <xsd:element ref="renderingType" minOccurs="0" maxOccurs="unbounded"/>
4870   </xsd:sequence>
4871 </xsd:complexType>
4872
4873 <!-- Single rendering element that contains rendering type (attribute) and
4874 data. -->
4875 <xsd:element name="rendering" type="tRendering"/>
4876 <xsd:complexType name="tRendering">
4877   <xsd:sequence>
4878     <xsd:any namespace="##other" processContents="lax" minOccurs="0"
4879 maxOccurs="unbounded"/>
4880   </xsd:sequence>
4881   <xsd:attribute name="type" type="xsd:QName" use="required"/>
4882 </xsd:complexType>
4883 <xsd:element name="renderings">
4884   <xsd:complexType>
4885     <xsd:sequence>
4886       <xsd:element ref="rendering" minOccurs="0" maxOccurs="unbounded"/>
4887     </xsd:sequence>
4888   </xsd:complexType>
4889 </xsd:element>
4890 <xsd:element name="description" type="xsd:string"/>
4891 <xsd:complexType name="tTaskInstanceData">
4892   <xsd:sequence>
4893     <!-- taskDetails contains task ID, meta data, presentation name and
4894 presentation subject. -->
4895     <xsd:element ref="taskDetails"/>
4896     <xsd:element ref="description"/>
4897     <xsd:element name="input" type="tMessagePartsData"/>

```

```

4898     <xsd:element name="output" type="tMessagePartsData" nillable="true"/>
4899     <xsd:element name="fault" type="tFaultData" nillable="true"
4900 minOccurs="0"/>
4901     <xsd:element ref="renderings" minOccurs="0"/>
4902     <xsd:element ref="comments" minOccurs="0"/>
4903     <xsd:element ref="attachmentInfos" minOccurs="0"/>
4904     <xsd:any namespace="##other" processContents="lax" minOccurs="0"
4905 maxOccurs="unbounded"/>
4906   </xsd:sequence>
4907 </xsd:complexType>
4908
4909 <!-- Defines the human task event types -->
4910 <xsd:simpleType name="tTaskEventType">
4911   <xsd:restriction base="xsd:string">
4912     <xsd:enumeration value="create"/>
4913     <xsd:enumeration value="claim"/>
4914     <xsd:enumeration value="start"/>
4915     <xsd:enumeration value="stop"/>
4916     <xsd:enumeration value="release"/>
4917     <xsd:enumeration value="suspend"/>
4918     <xsd:enumeration value="suspendUntil"/>
4919     <xsd:enumeration value="resume"/>
4920     <xsd:enumeration value="complete"/>
4921     <xsd:enumeration value="remove"/>
4922     <xsd:enumeration value="fail"/>
4923     <xsd:enumeration value="setPriority"/>
4924     <xsd:enumeration value="addAttachment"/>
4925     <xsd:enumeration value="deleteattachment"/>
4926     <xsd:enumeration value="addComment"/>
4927     <xsd:enumeration value="skip"/>
4928     <xsd:enumeration value="forward"/>
4929     <xsd:enumeration value="delegate"/>
4930     <xsd:enumeration value="setOutput"/>
4931     <xsd:enumeration value="deleteOutput"/>
4932     <xsd:enumeration value="setFault"/>
4933     <xsd:enumeration value="deleteFault"/>
4934     <xsd:enumeration value="activate"/>
4935     <xsd:enumeration value="nominate"/>
4936     <xsd:enumeration value="setGenericHumanRole"/>
4937     <xsd:enumeration value="expire"/>
4938     <xsd:enumeration value="escalated"/>
4939   </xsd:restriction>
4940 </xsd:simpleType>
4941 <xsd:element name="taskEvent">
4942   <xsd:complexType>
4943     <xsd:annotation>
4944       <xsd:documentation>
4945         A detailed event that represents a change in the task's state.
4946       </xsd:documentation>
4947     </xsd:annotation>
4948     <xsd:sequence>
4949       <!-- event id - unique per task -->
4950       <xsd:element name="id" type="xsd:integer"/>
4951       <!-- event date time -->
4952       <xsd:element name="eventTime" type="xsd:dateTime"/>
4953       <!-- task ID -->
4954       <xsd:element name="identifier" type="xsd:anyURI"/>

```

```

4955     <xsd:element name="principal" type="xsd:string" nillable="true"
4956 minOccurs="0"/>
4957     <!-- Event type. Note - using a restricted type limits extensibility
4958 to add custom event types. -->
4959     <xsd:element name="eventType" type="tTaskEventType"/>
4960     <!-- actual owner of the task before the event -->
4961     <xsd:element name="startOwner" type="xsd:string" nillable="true"
4962 minOccurs="0"/>
4963     <!-- actual owner of the task after the event -->
4964     <xsd:element name="endOwner" type="xsd:string" nillable="true"
4965 minOccurs="0"/>
4966     <!-- WSHT task status -->
4967     <xsd:element name="status" type="tStatus"/>
4968     <!-- boolean to indicate this event has optional data -->
4969     <xsd:element name="hasData" type="xsd:boolean" minOccurs="0"/>
4970     <xsd:element name="eventData" type="xsd:anyType" nillable="true"
4971 minOccurs="0"/>
4972     <xsd:element name="faultName" type="xsd:string" nillable="true"
4973 minOccurs="0"/>
4974     <!-- extensibility -->
4975     <xsd:any namespace="##other" processContents="lax" minOccurs="0"
4976 maxOccurs="unbounded"/>
4977     </xsd:sequence>
4978   </xsd:complexType>
4979 </xsd:element>
4980 <!-- Filter allow list event by eventId or other params such as status and
4981 event type -->
4982 <xsd:complexType name="tTaskHistoryFilter">
4983   <xsd:choice>
4984     <xsd:element name="eventId" type="xsd:integer"/>
4985     <!-- Filter to allow narrow down query by status, principal, event
4986 Type. -->
4987     <xsd:sequence>
4988       <xsd:element name="status" type="tStatus" minOccurs="0"
4989 maxOccurs="unbounded"/>
4990       <xsd:element name="eventType" type="tTaskEventType" minOccurs="0"
4991 maxOccurs="unbounded"/>
4992       <xsd:element name="principal" type="xsd:string" minOccurs="0"/>
4993       <xsd:element name="afterEventTime" type="xsd:dateTime"
4994 minOccurs="0"/>
4995       <xsd:element name="beforeEventTime" type="xsd:dateTime"
4996 minOccurs="0"/>
4997     </xsd:sequence>
4998   </xsd:choice>
4999 </xsd:complexType>
5000 </xsd:schema>

```

5001

D. WS-HumanTask Client API Port Type

```
5002 <?xml version="1.0" encoding="UTF-8"?>
5003 <!--
5004 Copyright (c) OASIS Open 2009. All Rights Reserved.
5005 -->
5006 <wsdl:definitions
5007     targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
5008 humantask/api/200803"
5009     xmlns="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/api/200803"
5010     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
5011     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
5012     xmlns:htt="http://docs.oasis-open.org/ns/bpel4people/ws-
5013 humantask/types/200803">
5014
5015     <wsdl:documentation>
5016         Web Service Definition for WS-HumanTask 1.1 - Operations for Client
5017 Applications
5018     </wsdl:documentation>
5019
5020     <wsdl:types>
5021         <xsd:schema
5022             targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
5023 humantask/api/200803"
5024             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
5025             xmlns:htt="http://docs.oasis-open.org/ns/bpel4people/ws-
5026 humantask/types/200803"
5027             elementFormDefault="qualified"
5028             blockDefault="#all">
5029
5030             <xsd:import
5031                 namespace="http://docs.oasis-open.org/ns/bpel4people/ws-
5032 humantask/types/200803"
5033                 schemaLocation="ws-humantask-types.xsd"/>
5034
5035             <!-- Input and output elements -->
5036             <xsd:element name="addAttachment">
5037                 <xsd:complexType>
5038                     <xsd:sequence>
5039                         <xsd:element name="taskIdIdentifier" type="xsd:anyURI"/>
5040                         <xsd:element name="name" type="xsd:string"/>
5041                         <xsd:element name="accessType" type="xsd:string"/>
5042                         <xsd:element name="contentType" type="xsd:string"/>
5043                         <xsd:element name="attachment" type="xsd:anyType"/>
5044                     </xsd:sequence>
5045                 </xsd:complexType>
5046             </xsd:element>
5047             <xsd:element name="addAttachmentResponse">
5048                 <xsd:complexType>
5049                     <xsd:sequence>
5050                         <xsd:element name="identifier" type="xsd:anyURI"/>
5051                     </xsd:sequence>
5052                 </xsd:complexType>
5053             </xsd:element>
5054
```

```

5055 <xsd:element name="addComment">
5056   <xsd:complexType>
5057     <xsd:sequence>
5058       <xsd:element name="identifier" type="xsd:anyURI"/>
5059       <xsd:element name="text" type="xsd:string"/>
5060     </xsd:sequence>
5061   </xsd:complexType>
5062 </xsd:element>
5063 <xsd:element name="addCommentResponse">
5064   <xsd:complexType>
5065     <xsd:sequence>
5066       <xsd:element name="commentID" type="xsd:anyURI"/>
5067     </xsd:sequence>
5068   </xsd:complexType>
5069 </xsd:element>
5070
5071 <xsd:element name="claim">
5072   <xsd:complexType>
5073     <xsd:sequence>
5074       <xsd:element name="identifier" type="xsd:anyURI"/>
5075     </xsd:sequence>
5076   </xsd:complexType>
5077 </xsd:element>
5078 <xsd:element name="claimResponse">
5079   <xsd:complexType>
5080     <xsd:sequence>
5081       <xsd:annotation>
5082         <xsd:documentation>Empty message</xsd:documentation>
5083       </xsd:annotation>
5084     </xsd:sequence>
5085   </xsd:complexType>
5086 </xsd:element>
5087
5088 <xsd:element name="batchClaim">
5089   <xsd:complexType>
5090     <xsd:sequence>
5091       <xsd:element name="identifier" type="xsd:anyURI"
5092 maxOccurs="unbounded"/>
5093     </xsd:sequence>
5094   </xsd:complexType>
5095 </xsd:element>
5096 <xsd:element name="batchClaimResponse">
5097   <xsd:complexType>
5098     <xsd:sequence>
5099       <xsd:element name="batchResponse" type="tBatchResponse"
5100 minOccurs="0" maxOccurs="unbounded"/>
5101     </xsd:sequence>
5102   </xsd:complexType>
5103 </xsd:element>
5104
5105 <xsd:element name="complete">
5106   <xsd:complexType>
5107     <xsd:sequence>
5108       <xsd:element name="identifier" type="xsd:anyURI"/>
5109       <xsd:element name="taskData" type="xsd:anyType" minOccurs="0"/>
5110     </xsd:sequence>
5111   </xsd:complexType>
5112 </xsd:element>

```

```

5113 <xsd:element name="completeResponse">
5114   <xsd:complexType>
5115     <xsd:sequence>
5116       <xsd:annotation>
5117         <xsd:documentation>Empty message</xsd:documentation>
5118       </xsd:annotation>
5119     </xsd:sequence>
5120   </xsd:complexType>
5121 </xsd:element>
5122
5123 <xsd:element name="batchComplete">
5124   <xsd:complexType>
5125     <xsd:sequence>
5126       <xsd:element name="identifier" type="xsd:anyURI"
5127 maxOccurs="unbounded"/>
5128       <xsd:element name="taskData" type="xsd:anyType" minOccurs="0"/>
5129     </xsd:sequence>
5130   </xsd:complexType>
5131 </xsd:element>
5132 <xsd:element name="batchCompleteResponse">
5133   <xsd:complexType>
5134     <xsd:sequence>
5135       <xsd:element name="batchResponse" type="tBatchResponse"
5136 minOccurs="0" maxOccurs="unbounded"/>
5137     </xsd:sequence>
5138   </xsd:complexType>
5139 </xsd:element>
5140
5141 <xsd:element name="delegate">
5142   <xsd:complexType>
5143     <xsd:sequence>
5144       <xsd:element name="identifier" type="xsd:anyURI"/>
5145       <xsd:element name="organizationalEntity"
5146 type="htt:tOrganizationalEntity"/>
5147     </xsd:sequence>
5148   </xsd:complexType>
5149 </xsd:element>
5150 <xsd:element name="delegateResponse">
5151   <xsd:complexType>
5152     <xsd:sequence>
5153       <xsd:annotation>
5154         <xsd:documentation>Empty message</xsd:documentation>
5155       </xsd:annotation>
5156     </xsd:sequence>
5157   </xsd:complexType>
5158 </xsd:element>
5159
5160 <xsd:element name="batchDelegate">
5161   <xsd:complexType>
5162     <xsd:sequence>
5163       <xsd:element name="identifier" type="xsd:anyURI"
5164 maxOccurs="unbounded"/>
5165       <xsd:element name="organizationalEntity"
5166 type="htt:tOrganizationalEntity"/>
5167     </xsd:sequence>
5168   </xsd:complexType>
5169 </xsd:element>
5170 <xsd:element name="batchDelegateResponse">

```

```

5171     <xsd:complexType>
5172     <xsd:sequence>
5173         <xsd:element name="batchResponse" type="tBatchResponse"
5174 minOccurs="0" maxOccurs="unbounded"/>
5175     </xsd:sequence>
5176 </xsd:complexType>
5177 </xsd:element>
5178
5179 <xsd:element name="deleteAttachment">
5180 <xsd:complexType>
5181 <xsd:sequence>
5182 <xsd:element name="taskIdentifier" type="xsd:anyURI"/>
5183 <xsd:element name="attachmentIdentifier" type="xsd:anyURI"/>
5184 </xsd:sequence>
5185 </xsd:complexType>
5186 </xsd:element>
5187 <xsd:element name="deleteAttachmentResponse">
5188 <xsd:complexType>
5189 <xsd:sequence>
5190 <xsd:annotation>
5191 <xsd:documentation>Empty message</xsd:documentation>
5192 </xsd:annotation>
5193 </xsd:sequence>
5194 </xsd:complexType>
5195 </xsd:element>
5196
5197 <xsd:element name="deleteComment">
5198 <xsd:complexType>
5199 <xsd:sequence>
5200 <xsd:element name="taskIdentifier" type="xsd:anyURI"/>
5201 <xsd:element name="commentIdentifier" type="xsd:anyURI"/>
5202 </xsd:sequence>
5203 </xsd:complexType>
5204 </xsd:element>
5205 <xsd:element name="deleteCommentResponse">
5206 <xsd:complexType>
5207 <xsd:sequence>
5208 <xsd:annotation>
5209 <xsd:documentation>Empty message</xsd:documentation>
5210 </xsd:annotation>
5211 </xsd:sequence>
5212 </xsd:complexType>
5213 </xsd:element>
5214
5215 <xsd:element name="deleteFault">
5216 <xsd:complexType>
5217 <xsd:sequence>
5218 <xsd:element name="identifier" type="xsd:anyURI"/>
5219 </xsd:sequence>
5220 </xsd:complexType>
5221 </xsd:element>
5222 <xsd:element name="deleteFaultResponse">
5223 <xsd:complexType>
5224 <xsd:sequence>
5225 <xsd:annotation>
5226 <xsd:documentation>Empty message</xsd:documentation>
5227 </xsd:annotation>
5228 </xsd:sequence>

```



```

5229     </xsd:complexType>
5230 </xsd:element>
5231
5232 <xsd:element name="deleteOutput">
5233   <xsd:complexType>
5234     <xsd:sequence>
5235       <xsd:element name="identifier" type="xsd:anyURI"/>
5236     </xsd:sequence>
5237   </xsd:complexType>
5238 </xsd:element>
5239 <xsd:element name="deleteOutputResponse">
5240   <xsd:complexType>
5241     <xsd:sequence>
5242       <xsd:annotation>
5243         <xsd:documentation>Empty message</xsd:documentation>
5244       </xsd:annotation>
5245     </xsd:sequence>
5246   </xsd:complexType>
5247 </xsd:element>
5248
5249 <xsd:element name="fail">
5250   <xsd:complexType>
5251     <xsd:sequence>
5252       <xsd:element name="identifier" type="xsd:anyURI"/>
5253       <xsd:element name="fault" type="htt:tFault" minOccurs="0"/>
5254     </xsd:sequence>
5255   </xsd:complexType>
5256 </xsd:element>
5257 <xsd:element name="failResponse">
5258   <xsd:complexType>
5259     <xsd:sequence>
5260       <xsd:annotation>
5261         <xsd:documentation>Empty message</xsd:documentation>
5262       </xsd:annotation>
5263     </xsd:sequence>
5264   </xsd:complexType>
5265 </xsd:element>
5266
5267 <xsd:element name="batchFail">
5268   <xsd:complexType>
5269     <xsd:sequence>
5270       <xsd:element name="identifier" type="xsd:anyURI"
5271 maxOccurs="unbounded"/>
5272       <xsd:element name="fault" type="htt:tFault" minOccurs="0"/>
5273     </xsd:sequence>
5274   </xsd:complexType>
5275 </xsd:element>
5276 <xsd:element name="batchFailResponse">
5277   <xsd:complexType>
5278     <xsd:sequence>
5279       <xsd:element name="batchResponse" type="tBatchResponse"
5280 minOccurs="0" maxOccurs="unbounded"/>
5281     </xsd:sequence>
5282   </xsd:complexType>
5283 </xsd:element>
5284
5285 <xsd:element name="forward">
5286   <xsd:complexType>

```



```

5287     <xsd:sequence>
5288         <xsd:element name="identifier" type="xsd:anyURI"/>
5289         <xsd:element name="organizationalEntity"
5290 type="htt:tOrganizationalEntity"/>
5291     </xsd:sequence>
5292 </xsd:complexType>
5293 </xsd:element>
5294 <xsd:element name="forwardResponse">
5295     <xsd:complexType>
5296         <xsd:sequence>
5297             <xsd:annotation>
5298                 <xsd:documentation>Empty message</xsd:documentation>
5299             </xsd:annotation>
5300         </xsd:sequence>
5301     </xsd:complexType>
5302 </xsd:element>
5303
5304     <xsd:element name="batchForward">
5305         <xsd:complexType>
5306             <xsd:sequence>
5307                 <xsd:element name="identifier" type="xsd:anyURI"
5308 maxOccurs="unbounded"/>
5309                 <xsd:element name="organizationalEntity"
5310 type="htt:tOrganizationalEntity"/>
5311             </xsd:sequence>
5312         </xsd:complexType>
5313     </xsd:element>
5314     <xsd:element name="batchForwardResponse">
5315         <xsd:complexType>
5316             <xsd:sequence>
5317                 <xsd:element name="batchResponse" type="tBatchResponse"
5318 minOccurs="0" maxOccurs="unbounded"/>
5319             </xsd:sequence>
5320         </xsd:complexType>
5321     </xsd:element>
5322
5323     <xsd:element name="getAttachment">
5324         <xsd:complexType>
5325             <xsd:sequence>
5326                 <xsd:element name="taskIdIdentifier" type="xsd:anyURI"/>
5327                 <xsd:element name="attachmentIdentifier" type="xsd:anyURI"/>
5328             </xsd:sequence>
5329         </xsd:complexType>
5330     </xsd:element>
5331     <xsd:element name="getAttachmentResponse">
5332         <xsd:complexType>
5333             <xsd:sequence>
5334                 <xsd:element name="attachment" type="htt:tAttachment"
5335 minOccurs="0" maxOccurs="unbounded"/>
5336             </xsd:sequence>
5337         </xsd:complexType>
5338     </xsd:element>
5339
5340     <xsd:element name="getAttachmentInfos">
5341         <xsd:complexType>
5342             <xsd:sequence>
5343                 <xsd:element name="identifier" type="xsd:anyURI"/>
5344             </xsd:sequence>

```

```

5345     </xsd:complexType>
5346 </xsd:element>
5347 <xsd:element name="getAttachmentInfosResponse">
5348   <xsd:complexType>
5349     <xsd:sequence>
5350       <xsd:element name="info" type="htt:tAttachmentInfo" minOccurs="0"
5351 maxOccurs="unbounded"/>
5352     </xsd:sequence>
5353   </xsd:complexType>
5354 </xsd:element>
5355
5356 <xsd:element name="getComments">
5357   <xsd:complexType>
5358     <xsd:sequence>
5359       <xsd:element name="identifier" type="xsd:anyURI"/>
5360     </xsd:sequence>
5361   </xsd:complexType>
5362 </xsd:element>
5363 <xsd:element name="getCommentsResponse">
5364   <xsd:complexType>
5365     <xsd:sequence>
5366       <xsd:element name="comment" type="htt:tComment" minOccurs="0"
5367 maxOccurs="unbounded"/>
5368     </xsd:sequence>
5369   </xsd:complexType>
5370 </xsd:element>
5371
5372 <xsd:element name="getFault">
5373   <xsd:complexType>
5374     <xsd:sequence>
5375       <xsd:element name="identifier" type="xsd:anyURI"/>
5376     </xsd:sequence>
5377   </xsd:complexType>
5378 </xsd:element>
5379 <xsd:element name="getFaultResponse">
5380   <xsd:complexType>
5381     <xsd:sequence>
5382       <xsd:element name="fault" type="htt:tFault"/>
5383     </xsd:sequence>
5384   </xsd:complexType>
5385 </xsd:element>
5386
5387 <xsd:element name="getInput">
5388   <xsd:complexType>
5389     <xsd:sequence>
5390       <xsd:element name="identifier" type="xsd:anyURI"/>
5391       <xsd:element name="part" type="xsd:NCName" minOccurs="0"/>
5392     </xsd:sequence>
5393   </xsd:complexType>
5394 </xsd:element>
5395 <xsd:element name="getInputResponse">
5396   <xsd:complexType>
5397     <xsd:sequence>
5398       <xsd:element name="taskData" type="xsd:anyType"/>
5399     </xsd:sequence>
5400   </xsd:complexType>
5401 </xsd:element>
5402

```

```

5403 <xsd:element name="getOutcome">
5404   <xsd:complexType>
5405     <xsd:sequence>
5406       <xsd:element name="identifier" type="xsd:anyURI"/>
5407     </xsd:sequence>
5408   </xsd:complexType>
5409 </xsd:element>
5410 <xsd:element name="getOutcomeResponse">
5411   <xsd:complexType>
5412     <xsd:sequence>
5413       <xsd:element name="outcome" type="xsd:string"/>
5414     </xsd:sequence>
5415   </xsd:complexType>
5416 </xsd:element>
5417
5418 <xsd:element name="getOutput">
5419   <xsd:complexType>
5420     <xsd:sequence>
5421       <xsd:element name="identifier" type="xsd:anyURI"/>
5422       <xsd:element name="part" type="xsd:NCName" minOccurs="0"/>
5423     </xsd:sequence>
5424   </xsd:complexType>
5425 </xsd:element>
5426 <xsd:element name="getOutputResponse">
5427   <xsd:complexType>
5428     <xsd:sequence>
5429       <xsd:element name="taskData" type="xsd:anyType"/>
5430     </xsd:sequence>
5431   </xsd:complexType>
5432 </xsd:element>
5433
5434 <xsd:element name="getParentTask">
5435   <xsd:complexType>
5436     <xsd:sequence>
5437       <xsd:element name="taskIdentifier" type="xsd:anyURI"/>
5438     </xsd:sequence>
5439   </xsd:complexType>
5440 </xsd:element>
5441 <xsd:element name="getParentTaskResponse">
5442   <xsd:complexType>
5443     <xsd:sequence>
5444       <xsd:element name="parentTask" type="htt:tTaskDetails"
5445 minOccurs="0"/>
5446     </xsd:sequence>
5447   </xsd:complexType>
5448 </xsd:element>
5449
5450 <xsd:element name="getParentTaskIdentifier">
5451   <xsd:complexType>
5452     <xsd:sequence>
5453       <xsd:element name="taskIdentifier" type="xsd:anyURI"/>
5454     </xsd:sequence>
5455   </xsd:complexType>
5456 </xsd:element>
5457 <xsd:element name="getParentTaskIdentifierResponse">
5458   <xsd:complexType>
5459     <xsd:sequence>

```

```

5460         <xsd:element name="parentTaskIdentifier" type="xsd:anyURI"
5461 minOccurs="0"/>
5462     </xsd:sequence>
5463 </xsd:complexType>
5464 </xsd:element>
5465
5466 <xsd:element name="getRendering">
5467     <xsd:complexType>
5468         <xsd:sequence>
5469             <xsd:element name="identifier" type="xsd:anyURI"/>
5470             <xsd:element name="renderingType" type="xsd:QName"/>
5471         </xsd:sequence>
5472     </xsd:complexType>
5473 </xsd:element>
5474 <xsd:element name="getRenderingResponse">
5475     <xsd:complexType>
5476         <xsd:sequence>
5477             <xsd:element name="rendering" type="xsd:anyType"/>
5478         </xsd:sequence>
5479     </xsd:complexType>
5480 </xsd:element>
5481
5482 <xsd:element name="getRenderingTypes">
5483     <xsd:complexType>
5484         <xsd:sequence>
5485             <xsd:element name="identifier" type="xsd:anyURI"/>
5486         </xsd:sequence>
5487     </xsd:complexType>
5488 </xsd:element>
5489 <xsd:element name="getRenderingTypesResponse">
5490     <xsd:complexType>
5491         <xsd:sequence>
5492             <xsd:element name="renderingType" type="xsd:QName" minOccurs="0"
5493 maxOccurs="unbounded"/>
5494         </xsd:sequence>
5495     </xsd:complexType>
5496 </xsd:element>
5497
5498 <xsd:element name="getSubtaskIdentifiers">
5499     <xsd:complexType>
5500         <xsd:sequence>
5501             <xsd:element name="taskIdentifier" type="xsd:anyURI"/>
5502         </xsd:sequence>
5503     </xsd:complexType>
5504 </xsd:element>
5505 <xsd:element name="getSubtaskIdentifiersResponse">
5506     <xsd:complexType>
5507         <xsd:sequence>
5508             <xsd:element name="subtaskIdentifier" type="xsd:anyURI"
5509 minOccurs="0" maxOccurs="unbounded"/>
5510         </xsd:sequence>
5511     </xsd:complexType>
5512 </xsd:element>
5513
5514 <xsd:element name="getSubtasks">
5515     <xsd:complexType>
5516         <xsd:sequence>
5517             <xsd:element name="taskIdentifier" type="xsd:anyURI"/>

```

```

5518     </xsd:sequence>
5519   </xsd:complexType>
5520 </xsd:element>
5521 <xsd:element name="getSubtasksResponse">
5522   <xsd:complexType>
5523     <xsd:sequence>
5524       <xsd:element name="subtask" type="htt:tTaskDetails" minOccurs="0"
5525 maxOccurs="unbounded"/>
5526     </xsd:sequence>
5527   </xsd:complexType>
5528 </xsd:element>
5529
5530 <xsd:element name="getTaskDescription">
5531   <xsd:complexType>
5532     <xsd:sequence>
5533       <xsd:element name="identifier" type="xsd:anyURI"/>
5534       <xsd:element name="contentType" type="xsd:string" minOccurs="0"/>
5535     </xsd:sequence>
5536   </xsd:complexType>
5537 </xsd:element>
5538 <xsd:element name="getTaskDescriptionResponse">
5539   <xsd:complexType>
5540     <xsd:sequence>
5541       <xsd:element name="description" type="xsd:string"/>
5542     </xsd:sequence>
5543   </xsd:complexType>
5544 </xsd:element>
5545
5546 <xsd:element name="getTaskDetails">
5547   <xsd:complexType>
5548     <xsd:sequence>
5549       <xsd:element name="identifier" type="xsd:anyURI"/>
5550     </xsd:sequence>
5551   </xsd:complexType>
5552 </xsd:element>
5553 <xsd:element name="getTaskDetailsResponse">
5554   <xsd:complexType>
5555     <xsd:sequence>
5556       <xsd:element name="taskDetails" type="htt:tTaskDetails"/>
5557     </xsd:sequence>
5558   </xsd:complexType>
5559 </xsd:element>
5560
5561 <xsd:element name="getTaskHistory">
5562   <xsd:complexType>
5563     <xsd:sequence>
5564       <xsd:element name="identifier" type="xsd:anyURI"/>
5565       <xsd:element name="filter" type="htt:tTaskHistoryFilter"
5566 minOccurs="0"/>
5567       <xsd:element name="startIndex" type="xsd:int" minOccurs="0"/>
5568       <xsd:element name="maxTasks" type="xsd:int" minOccurs="0"/>
5569     </xsd:sequence>
5570     <xsd:attribute name="includeData" type="xsd:boolean"/>
5571   </xsd:complexType>
5572 </xsd:element>
5573 <xsd:element name="getTaskHistoryResponse">
5574   <xsd:complexType>
5575     <xsd:sequence>

```

```

5576         <xsd:element name="taskEvent" type="htt:tTaskEventType"
5577 minOccurs="0" maxOccurs="unbounded"/>
5578         </xsd:sequence>
5579     </xsd:complexType>
5580 </xsd:element>
5581
5582     <xsd:element name="getTaskInstanceData">
5583         <xsd:complexType>
5584             <xsd:sequence>
5585                 <xsd:element name="identifier" type="xsd:anyURI"/>
5586                 <xsd:element name="properties" type="xsd:string"/>
5587                 <xsd:element name="renderingPreferences"
5588 type="htt:tRenderingTypes" minOccurs="0" maxOccurs="unbounded"/>
5589             </xsd:sequence>
5590         </xsd:complexType>
5591     </xsd:element>
5592     <xsd:element name="getTaskInstanceDataResponse">
5593         <xsd:complexType>
5594             <xsd:sequence>
5595                 <xsd:element name="taskInstanceData"
5596 type="htt:tTaskInstanceData"/>
5597             </xsd:sequence>
5598         </xsd:complexType>
5599     </xsd:element>
5600
5601     <xsd:element name="getTaskOperations">
5602         <xsd:complexType>
5603             <xsd:sequence>
5604                 <xsd:element name="identifier" type="xsd:anyURI"/>
5605             </xsd:sequence>
5606         </xsd:complexType>
5607     </xsd:element>
5608     <xsd:element name="getTaskOperationsResponse">
5609         <xsd:complexType>
5610             <xsd:sequence>
5611                 <xsd:element name="taskOperations" type="htt:tTaskOperations"/>
5612             </xsd:sequence>
5613         </xsd:complexType>
5614     </xsd:element>
5615
5616     <xsd:element name="hasSubtasks">
5617         <xsd:complexType>
5618             <xsd:sequence>
5619                 <xsd:element name="taskIdentifier" type="xsd:anyURI"/>
5620             </xsd:sequence>
5621         </xsd:complexType>
5622     </xsd:element>
5623     <xsd:element name="hasSubtasksResponse">
5624         <xsd:complexType>
5625             <xsd:sequence>
5626                 <xsd:element name="result" type="xsd:boolean"/>
5627             </xsd:sequence>
5628         </xsd:complexType>
5629     </xsd:element>
5630
5631     <xsd:element name="instantiateSubtask">
5632         <xsd:complexType>
5633             <xsd:sequence>

```

```

5634         <xsd:element name="taskIdentifier" type="xsd:anyURI"/>
5635         <xsd:element name="name" type="xsd:string"/>
5636     </xsd:sequence>
5637 </xsd:complexType>
5638 </xsd:element>
5639 <xsd:element name="instantiateSubtaskResponse">
5640     <xsd:complexType>
5641     <xsd:sequence>
5642         <xsd:element name="subtaskIdentifier" type="xsd:anyURI"/>
5643     </xsd:sequence>
5644 </xsd:complexType>
5645 </xsd:element>
5646
5647 <xsd:element name="isSubtask">
5648     <xsd:complexType>
5649     <xsd:sequence>
5650         <xsd:element name="taskIdentifier" type="xsd:anyURI"/>
5651     </xsd:sequence>
5652 </xsd:complexType>
5653 </xsd:element>
5654 <xsd:element name="isSubtaskResponse">
5655     <xsd:complexType>
5656     <xsd:sequence>
5657         <xsd:element name="result" type="xsd:boolean"/>
5658     </xsd:sequence>
5659 </xsd:complexType>
5660 </xsd:element>
5661
5662 <xsd:element name="release">
5663     <xsd:complexType>
5664     <xsd:sequence>
5665         <xsd:element name="identifier" type="xsd:anyURI"/>
5666     </xsd:sequence>
5667 </xsd:complexType>
5668 </xsd:element>
5669 <xsd:element name="releaseResponse">
5670     <xsd:complexType>
5671     <xsd:sequence>
5672         <xsd:annotation>
5673             <xsd:documentation>Empty message</xsd:documentation>
5674         </xsd:annotation>
5675     </xsd:sequence>
5676 </xsd:complexType>
5677 </xsd:element>
5678
5679 <xsd:element name="batchRelease">
5680     <xsd:complexType>
5681     <xsd:sequence>
5682         <xsd:element name="identifier" type="xsd:anyURI"
5683 maxOccurs="unbounded"/>
5684     </xsd:sequence>
5685 </xsd:complexType>
5686 </xsd:element>
5687 <xsd:element name="batchReleaseResponse">
5688     <xsd:complexType>
5689     <xsd:sequence>
5690         <xsd:element name="batchResponse" type="tBatchResponse"
5691 minOccurs="0" maxOccurs="unbounded"/>

```

```

5692     </xsd:sequence>
5693   </xsd:complexType>
5694 </xsd:element>
5695
5696   <xsd:element name="remove">
5697     <xsd:complexType>
5698       <xsd:sequence>
5699         <xsd:element name="identifier" type="xsd:anyURI"/>
5700       </xsd:sequence>
5701     </xsd:complexType>
5702   </xsd:element>
5703 <xsd:element name="removeResponse">
5704   <xsd:complexType>
5705     <xsd:sequence>
5706       <xsd:annotation>
5707         <xsd:documentation>Empty message</xsd:documentation>
5708       </xsd:annotation>
5709     </xsd:sequence>
5710   </xsd:complexType>
5711 </xsd:element>
5712
5713   <xsd:element name="batchRemove">
5714     <xsd:complexType>
5715       <xsd:sequence>
5716         <xsd:element name="identifier" type="xsd:anyURI"
5717 maxOccurs="unbounded"/>
5718       </xsd:sequence>
5719     </xsd:complexType>
5720   </xsd:element>
5721 <xsd:element name="batchRemoveResponse">
5722   <xsd:complexType>
5723     <xsd:sequence>
5724       <xsd:element name="batchResponse" type="tBatchResponse"
5725 minOccurs="0" maxOccurs="unbounded"/>
5726     </xsd:sequence>
5727   </xsd:complexType>
5728 </xsd:element>
5729
5730   <xsd:element name="resume">
5731     <xsd:complexType>
5732       <xsd:sequence>
5733         <xsd:element name="identifier" type="xsd:anyURI"/>
5734       </xsd:sequence>
5735     </xsd:complexType>
5736   </xsd:element>
5737 <xsd:element name="resumeResponse">
5738   <xsd:complexType>
5739     <xsd:sequence>
5740       <xsd:annotation>
5741         <xsd:documentation>Empty message</xsd:documentation>
5742       </xsd:annotation>
5743     </xsd:sequence>
5744   </xsd:complexType>
5745 </xsd:element>
5746
5747   <xsd:element name="batchResume">
5748     <xsd:complexType>
5749     <xsd:sequence>

```



```

5750         <xsd:element name="identifier" type="xsd:anyURI"
5751 maxOccurs="unbounded"/>
5752     </xsd:sequence>
5753 </xsd:complexType>
5754 </xsd:element>
5755 <xsd:element name="batchResumeResponse">
5756 <xsd:complexType>
5757 <xsd:sequence>
5758 <xsd:element name="batchResponse" type="tBatchResponse"
5759 minOccurs="0" maxOccurs="unbounded"/>
5760 </xsd:sequence>
5761 </xsd:complexType>
5762 </xsd:element>
5763
5764 <xsd:element name="setFault">
5765 <xsd:complexType>
5766 <xsd:sequence>
5767 <xsd:element name="identifier" type="xsd:anyURI"/>
5768 <xsd:element name="fault" type="htt:tFault"/>
5769 </xsd:sequence>
5770 </xsd:complexType>
5771 </xsd:element>
5772 <xsd:element name="setFaultResponse">
5773 <xsd:complexType>
5774 <xsd:sequence>
5775 <xsd:annotation>
5776 <xsd:documentation>Empty message</xsd:documentation>
5777 </xsd:annotation>
5778 </xsd:sequence>
5779 </xsd:complexType>
5780 </xsd:element>
5781
5782 <xsd:element name="setOutput">
5783 <xsd:complexType>
5784 <xsd:sequence>
5785 <xsd:element name="identifier" type="xsd:anyURI"/>
5786 <xsd:element name="part" type="xsd:NCName" minOccurs="0"/>
5787 <xsd:element name="taskData" type="xsd:anyType"/>
5788 </xsd:sequence>
5789 </xsd:complexType>
5790 </xsd:element>
5791 <xsd:element name="setOutputResponse">
5792 <xsd:complexType>
5793 <xsd:sequence>
5794 <xsd:annotation>
5795 <xsd:documentation>Empty message</xsd:documentation>
5796 </xsd:annotation>
5797 </xsd:sequence>
5798 </xsd:complexType>
5799 </xsd:element>
5800
5801 <xsd:element name="setPriority">
5802 <xsd:complexType>
5803 <xsd:sequence>
5804 <xsd:element name="identifier" type="xsd:anyURI"/>
5805 <xsd:element name="priority" type="htt:tPriority"/>
5806 </xsd:sequence>
5807 </xsd:complexType>

```

```

5808 </xsd:element>
5809 <xsd:element name="setPriorityResponse">
5810 <xsd:complexType>
5811 <xsd:sequence>
5812 <xsd:annotation>
5813 <xsd:documentation>Empty message</xsd:documentation>
5814 </xsd:annotation>
5815 </xsd:sequence>
5816 </xsd:complexType>
5817 </xsd:element>
5818
5819 <xsd:element name="batchSetPriority">
5820 <xsd:complexType>
5821 <xsd:sequence>
5822 <xsd:element name="identifier" type="xsd:anyURI"
5823 maxOccurs="unbounded"/>
5824 <xsd:element name="priority" type="htt:tPriority"/>
5825 </xsd:sequence>
5826 </xsd:complexType>
5827 </xsd:element>
5828 <xsd:element name="batchSetPriorityResponse">
5829 <xsd:complexType>
5830 <xsd:sequence>
5831 <xsd:element name="batchResponse" type="tBatchResponse"
5832 minOccurs="0" maxOccurs="unbounded"/>
5833 </xsd:sequence>
5834 </xsd:complexType>
5835 </xsd:element>
5836
5837 <xsd:element name="setTaskCompletionDeadlineExpression">
5838 <xsd:complexType>
5839 <xsd:sequence>
5840 <xsd:element name="identifier" type="xsd:anyURI"/>
5841 <xsd:element name="deadlineName" type="xsd:NCName"/>
5842 <xsd:element name="deadlineExpression" type="xsd:string"/>
5843 </xsd:sequence>
5844 </xsd:complexType>
5845 </xsd:element>
5846 <xsd:element name="setTaskCompletionDeadlineExpressionResponse">
5847 <xsd:complexType>
5848 <xsd:sequence>
5849 <xsd:annotation>
5850 <xsd:documentation>Empty message</xsd:documentation>
5851 </xsd:annotation>
5852 </xsd:sequence>
5853 </xsd:complexType>
5854 </xsd:element>
5855
5856 <xsd:element name="setTaskCompletionDurationExpression">
5857 <xsd:complexType>
5858 <xsd:sequence>
5859 <xsd:element name="identifier" type="xsd:anyURI"/>
5860 <xsd:element name="deadlineName" type="xsd:NCName"/>
5861 <xsd:element name="durationExpression" type="xsd:string"/>
5862 </xsd:sequence>
5863 </xsd:complexType>
5864 </xsd:element>
5865 <xsd:element name="setTaskCompletionDurationExpressionResponse">

```

```

5866     <xsd:complexType>
5867         <xsd:sequence>
5868             <xsd:annotation>
5869                 <xsd:documentation>Empty message</xsd:documentation>
5870             </xsd:annotation>
5871         </xsd:sequence>
5872     </xsd:complexType>
5873 </xsd:element>
5874
5875 <xsd:element name="setTaskStartDeadlineExpression">
5876     <xsd:complexType>
5877         <xsd:sequence>
5878             <xsd:element name="identifier" type="xsd:anyURI"/>
5879             <xsd:element name="deadlineName" type="xsd:NCName"/>
5880             <xsd:element name="deadlineExpression" type="xsd:string"/>
5881         </xsd:sequence>
5882     </xsd:complexType>
5883 </xsd:element>
5884 <xsd:element name="setTaskStartDeadlineExpressionResponse">
5885     <xsd:complexType>
5886         <xsd:sequence>
5887             <xsd:annotation>
5888                 <xsd:documentation>Empty message</xsd:documentation>
5889             </xsd:annotation>
5890         </xsd:sequence>
5891     </xsd:complexType>
5892 </xsd:element>
5893
5894 <xsd:element name="setTaskStartDurationExpression">
5895     <xsd:complexType>
5896         <xsd:sequence>
5897             <xsd:element name="identifier" type="xsd:anyURI"/>
5898             <xsd:element name="deadlineName" type="xsd:NCName"/>
5899             <xsd:element name="durationExpression" type="xsd:string"/>
5900         </xsd:sequence>
5901     </xsd:complexType>
5902 </xsd:element>
5903 <xsd:element name="setTaskStartDurationExpressionResponse">
5904     <xsd:complexType>
5905         <xsd:sequence>
5906             <xsd:annotation>
5907                 <xsd:documentation>Empty message</xsd:documentation>
5908             </xsd:annotation>
5909         </xsd:sequence>
5910     </xsd:complexType>
5911 </xsd:element>
5912
5913 <xsd:element name="skip">
5914     <xsd:complexType>
5915         <xsd:sequence>
5916             <xsd:element name="identifier" type="xsd:anyURI"/>
5917         </xsd:sequence>
5918     </xsd:complexType>
5919 </xsd:element>
5920 <xsd:element name="skipResponse">
5921     <xsd:complexType>
5922         <xsd:sequence>
5923             <xsd:annotation>

```

```

5924         <xsd:documentation>Empty message</xsd:documentation>
5925     </xsd:annotation>
5926 </xsd:sequence>
5927 </xsd:complexType>
5928 </xsd:element>
5929
5930 <xsd:element name="batchSkip">
5931 <xsd:complexType>
5932 <xsd:sequence>
5933 <xsd:element name="identifier" type="xsd:anyURI"
5934 maxOccurs="unbounded"/>
5935 </xsd:sequence>
5936 </xsd:complexType>
5937 </xsd:element>
5938 <xsd:element name="batchSkipResponse">
5939 <xsd:complexType>
5940 <xsd:sequence>
5941 <xsd:element name="batchResponse" type="tBatchResponse"
5942 minOccurs="0" maxOccurs="unbounded"/>
5943 </xsd:sequence>
5944 </xsd:complexType>
5945 </xsd:element>
5946
5947 <xsd:element name="start">
5948 <xsd:complexType>
5949 <xsd:sequence>
5950 <xsd:element name="identifier" type="xsd:anyURI"/>
5951 </xsd:sequence>
5952 </xsd:complexType>
5953 </xsd:element>
5954 <xsd:element name="startResponse">
5955 <xsd:complexType>
5956 <xsd:sequence>
5957 <xsd:annotation>
5958 <xsd:documentation>Empty message</xsd:documentation>
5959 </xsd:annotation>
5960 </xsd:sequence>
5961 </xsd:complexType>
5962 </xsd:element>
5963
5964 <xsd:element name="batchStart">
5965 <xsd:complexType>
5966 <xsd:sequence>
5967 <xsd:element name="identifier" type="xsd:anyURI"
5968 maxOccurs="unbounded"/>
5969 </xsd:sequence>
5970 </xsd:complexType>
5971 </xsd:element>
5972 <xsd:element name="batchStartResponse">
5973 <xsd:complexType>
5974 <xsd:sequence>
5975 <xsd:element name="batchResponse" type="tBatchResponse"
5976 minOccurs="0" maxOccurs="unbounded"/>
5977 </xsd:sequence>
5978 </xsd:complexType>
5979 </xsd:element>
5980
5981 <xsd:element name="stop">

```

```

5982     <xsd:complexType>
5983         <xsd:sequence>
5984             <xsd:element name="identifier" type="xsd:anyURI"/>
5985         </xsd:sequence>
5986     </xsd:complexType>
5987 </xsd:element>
5988 <xsd:element name="stopResponse">
5989     <xsd:complexType>
5990         <xsd:sequence>
5991             <xsd:annotation>
5992                 <xsd:documentation>Empty message</xsd:documentation>
5993             </xsd:annotation>
5994         </xsd:sequence>
5995     </xsd:complexType>
5996 </xsd:element>
5997
5998     <xsd:element name="batchStop">
5999         <xsd:complexType>
6000             <xsd:sequence>
6001                 <xsd:element name="identifier" type="xsd:anyURI"
6002 maxOccurs="unbounded"/>
6003             </xsd:sequence>
6004         </xsd:complexType>
6005     </xsd:element>
6006     <xsd:element name="batchStopResponse">
6007         <xsd:complexType>
6008             <xsd:sequence>
6009                 <xsd:element name="batchResponse" type="tBatchResponse"
6010 minOccurs="0" maxOccurs="unbounded"/>
6011             </xsd:sequence>
6012         </xsd:complexType>
6013     </xsd:element>
6014
6015     <xsd:element name="suspend">
6016         <xsd:complexType>
6017             <xsd:sequence>
6018                 <xsd:element name="identifier" type="xsd:anyURI"/>
6019             </xsd:sequence>
6020         </xsd:complexType>
6021     </xsd:element>
6022     <xsd:element name="suspendResponse">
6023         <xsd:complexType>
6024             <xsd:sequence>
6025                 <xsd:annotation>
6026                     <xsd:documentation>Empty message</xsd:documentation>
6027                 </xsd:annotation>
6028             </xsd:sequence>
6029         </xsd:complexType>
6030     </xsd:element>
6031
6032     <xsd:element name="batchSuspend">
6033         <xsd:complexType>
6034             <xsd:sequence>
6035                 <xsd:element name="identifier" type="xsd:anyURI"
6036 maxOccurs="unbounded"/>
6037             </xsd:sequence>
6038         </xsd:complexType>
6039     </xsd:element>

```

```

6040     <xsd:element name="batchSuspendResponse">
6041         <xsd:complexType>
6042             <xsd:sequence>
6043                 <xsd:element name="batchResponse" type="tBatchResponse"
6044 minOccurs="0" maxOccurs="unbounded"/>
6045             </xsd:sequence>
6046         </xsd:complexType>
6047     </xsd:element>
6048
6049     <xsd:element name="suspendUntil">
6050         <xsd:complexType>
6051             <xsd:sequence>
6052                 <xsd:element name="identifier" type="xsd:anyURI"/>
6053                 <xsd:element name="time" type="htt:tTime"/>
6054             </xsd:sequence>
6055         </xsd:complexType>
6056     </xsd:element>
6057     <xsd:element name="suspendUntilResponse">
6058         <xsd:complexType>
6059             <xsd:sequence>
6060                 <xsd:annotation>
6061                     <xsd:documentation>Empty message</xsd:documentation>
6062                 </xsd:annotation>
6063             </xsd:sequence>
6064         </xsd:complexType>
6065     </xsd:element>
6066
6067     <xsd:element name="batchSuspendUntil">
6068         <xsd:complexType>
6069             <xsd:sequence>
6070                 <xsd:element name="identifier" type="xsd:anyURI"
6071 maxOccurs="unbounded"/>
6072                 <xsd:element name="time" type="htt:tTime"/>
6073             </xsd:sequence>
6074         </xsd:complexType>
6075     </xsd:element>
6076     <xsd:element name="batchSuspendUntilResponse">
6077         <xsd:complexType>
6078             <xsd:sequence>
6079                 <xsd:element name="batchResponse" type="tBatchResponse"
6080 minOccurs="0" maxOccurs="unbounded"/>
6081             </xsd:sequence>
6082         </xsd:complexType>
6083     </xsd:element>
6084
6085     <xsd:element name="updateComment">
6086         <xsd:complexType>
6087             <xsd:sequence>
6088                 <xsd:element name="taskIdIdentifier" type="xsd:anyURI"/>
6089                 <xsd:element name="commentIdentifier" type="xsd:anyURI"/>
6090                 <xsd:element name="text" type="xsd:string"/>
6091             </xsd:sequence>
6092         </xsd:complexType>
6093     </xsd:element>
6094     <xsd:element name="updateCommentResponse">
6095         <xsd:complexType>
6096             <xsd:sequence>
6097                 <xsd:annotation>

```

```

6098         <xsd:documentation>Empty message</xsd:documentation>
6099     </xsd:annotation>
6100 </xsd:sequence>
6101 </xsd:complexType>
6102 </xsd:element>
6103
6104     <xsd:element name="getMyTaskAbstracts">
6105         <xsd:complexType>
6106             <xsd:sequence>
6107                 <xsd:element name="taskType" type="xsd:string"/>
6108                 <xsd:element name="genericHumanRole" type="xsd:string"
6109 minOccurs="0"/>
6110                 <xsd:element name="workQueue" type="xsd:string" minOccurs="0"/>
6111                 <xsd:element name="status" type="htt:tStatus" minOccurs="0"
6112 maxOccurs="unbounded"/>
6113                 <xsd:element name="whereClause" type="xsd:string" minOccurs="0"/>
6114                 <xsd:element name="orderByClause" type="xsd:string"
6115 minOccurs="0"/>
6116                 <xsd:element name="createdOnClause" type="xsd:string"
6117 minOccurs="0"/>
6118                 <xsd:element name="maxTasks" type="xsd:int" minOccurs="0"/>
6119                 <xsd:element name="taskIndexOffset" type="xsd:int"
6120 minOccurs="0"/>
6121             </xsd:sequence>
6122         </xsd:complexType>
6123     </xsd:element>
6124     <xsd:element name="getMyTaskAbstractsResponse">
6125         <xsd:complexType>
6126             <xsd:sequence>
6127                 <xsd:element name="taskAbstract" type="htt:tTaskAbstract"
6128 minOccurs="0" maxOccurs="unbounded"/>
6129             </xsd:sequence>
6130         </xsd:complexType>
6131     </xsd:element>
6132
6133     <xsd:element name="getMyTaskDetails">
6134         <xsd:complexType>
6135             <xsd:sequence>
6136                 <xsd:element name="taskType" type="xsd:string"/>
6137                 <xsd:element name="genericHumanRole" type="xsd:string"
6138 minOccurs="0"/>
6139                 <xsd:element name="workQueue" type="xsd:string" minOccurs="0"/>
6140                 <xsd:element name="status" type="htt:tStatus" minOccurs="0"
6141 maxOccurs="unbounded"/>
6142                 <xsd:element name="whereClause" type="xsd:string" minOccurs="0"/>
6143                 <xsd:element name="orderByClause" type="xsd:string"
6144 minOccurs="0"/>
6145                 <xsd:element name="createdOnClause" type="xsd:string"
6146 minOccurs="0"/>
6147                 <xsd:element name="maxTasks" type="xsd:int" minOccurs="0"/>
6148                 <xsd:element name="taskIndexOffset" type="xsd:int"
6149 minOccurs="0"/>
6150             </xsd:sequence>
6151         </xsd:complexType>
6152     </xsd:element>
6153     <xsd:element name="getMyTaskDetailsResponse">
6154         <xsd:complexType>
6155             <xsd:sequence>

```

```

6156         <xsd:element name="taskDetails" type="htt:tTaskDetails"
6157 minOccurs="0" maxOccurs="unbounded"/>
6158         </xsd:sequence>
6159     </xsd:complexType>
6160 </xsd:element>
6161
6162     <xsd:element name="query">
6163         <xsd:complexType>
6164             <xsd:sequence>
6165                 <xsd:element name="selectClause" type="xsd:string"/>
6166                 <xsd:element name="whereClause" type="xsd:string" minOccurs="0"/>
6167                 <xsd:element name="orderByClause" type="xsd:string"
6168 minOccurs="0"/>
6169                 <xsd:element name="maxTasks" type="xsd:int" minOccurs="0"/>
6170                 <xsd:element name="taskIndexOffset" type="xsd:int"
6171 minOccurs="0"/>
6172             </xsd:sequence>
6173         </xsd:complexType>
6174     </xsd:element>
6175     <xsd:element name="queryResponse">
6176         <xsd:complexType>
6177             <xsd:sequence>
6178                 <xsd:element name="taskQueryResultSet"
6179 type="htt:tTaskQueryResultSet"/>
6180             </xsd:sequence>
6181         </xsd:complexType>
6182     </xsd:element>
6183
6184     <xsd:element name="activate">
6185         <xsd:complexType>
6186             <xsd:sequence>
6187                 <xsd:element name="identifier" type="xsd:anyURI"/>
6188             </xsd:sequence>
6189         </xsd:complexType>
6190     </xsd:element>
6191     <xsd:element name="activateResponse">
6192         <xsd:complexType>
6193             <xsd:sequence>
6194                 <xsd:annotation>
6195                     <xsd:documentation>Empty message</xsd:documentation>
6196                 </xsd:annotation>
6197             </xsd:sequence>
6198         </xsd:complexType>
6199     </xsd:element>
6200
6201     <xsd:element name="batchActivate">
6202         <xsd:complexType>
6203             <xsd:sequence>
6204                 <xsd:element name="identifier" type="xsd:anyURI"
6205 maxOccurs="unbounded"/>
6206             </xsd:sequence>
6207         </xsd:complexType>
6208     </xsd:element>
6209     <xsd:element name="batchActivateResponse">
6210         <xsd:complexType>
6211             <xsd:sequence>
6212                 <xsd:element name="batchResponse" type="tBatchResponse"
6213 minOccurs="0" maxOccurs="unbounded"/>

```



```

6214     </xsd:sequence>
6215     </xsd:complexType>
6216 </xsd:element>
6217
6218     <xsd:element name="nominate">
6219         <xsd:complexType>
6220             <xsd:sequence>
6221                 <xsd:element name="identifier" type="xsd:anyURI"/>
6222                 <xsd:element name="organizationalEntity"
6223 type="http:OrganizationalEntity"/>
6224             </xsd:sequence>
6225         </xsd:complexType>
6226 </xsd:element>
6227 <xsd:element name="nominateResponse">
6228     <xsd:complexType>
6229         <xsd:sequence>
6230             <xsd:annotation>
6231                 <xsd:documentation>Empty message</xsd:documentation>
6232             </xsd:annotation>
6233         </xsd:sequence>
6234     </xsd:complexType>
6235 </xsd:element>
6236
6237     <xsd:element name="batchNominate">
6238         <xsd:complexType>
6239             <xsd:sequence>
6240                 <xsd:element name="identifier" type="xsd:anyURI"
6241 maxOccurs="unbounded"/>
6242             </xsd:sequence>
6243         </xsd:complexType>
6244 </xsd:element>
6245 <xsd:element name="batchNominateResponse">
6246     <xsd:complexType>
6247         <xsd:sequence>
6248             <xsd:element name="batchResponse" type="tBatchResponse"
6249 minOccurs="0" maxOccurs="unbounded"/>
6250         </xsd:sequence>
6251     </xsd:complexType>
6252 </xsd:element>
6253
6254     <xsd:element name="setGenericHumanRole">
6255         <xsd:complexType>
6256             <xsd:sequence>
6257                 <xsd:element name="identifier" type="xsd:anyURI"/>
6258                 <xsd:element name="genericHumanRole" type="xsd:string"/>
6259                 <xsd:element name="organizationalEntity"
6260 type="http:OrganizationalEntity"/>
6261             </xsd:sequence>
6262         </xsd:complexType>
6263 </xsd:element>
6264 <xsd:element name="setGenericHumanRoleResponse">
6265     <xsd:complexType>
6266         <xsd:sequence>
6267             <xsd:annotation>
6268                 <xsd:documentation>Empty message</xsd:documentation>
6269             </xsd:annotation>
6270         </xsd:sequence>
6271     </xsd:complexType>

```

```

6272     </xsd:element>
6273
6274     <xsd:element name="batchSetGenericHumanRole">
6275         <xsd:complexType>
6276             <xsd:sequence>
6277                 <xsd:element name="identifier" type="xsd:anyURI"
6278 maxOccurs="unbounded"/>
6279                 <xsd:element name="genericHumanRole" type="xsd:string"/>
6280                 <xsd:element name="organizationalEntity"
6281 type="http:OrganizationalEntity"/>
6282             </xsd:sequence>
6283         </xsd:complexType>
6284     </xsd:element>
6285     <xsd:element name="batchSetGenericHumanRoleResponse">
6286         <xsd:complexType>
6287             <xsd:sequence>
6288                 <xsd:element name="batchResponse" type="tBatchResponse"
6289 minOccurs="0" maxOccurs="unbounded"/>
6290             </xsd:sequence>
6291         </xsd:complexType>
6292     </xsd:element>
6293
6294     <!-- Fault elements -->
6295     <xsd:element name="illegalState">
6296         <xsd:complexType>
6297             <xsd:sequence>
6298                 <xsd:element name="status" type="http:tStatus"/>
6299                 <xsd:element name="message" type="xsd:string"/>
6300             </xsd:sequence>
6301         </xsd:complexType>
6302     </xsd:element>
6303
6304     <xsd:element name="illegalArgument" type="xsd:string"/>
6305
6306     <xsd:element name="illegalAccess" type="xsd:string"/>
6307
6308     <xsd:element name="illegalOperation" type="xsd:string"/>
6309
6310     <xsd:element name="recipientNotAllowed" type="xsd:string"/>
6311
6312     <xsd:complexType name="tBatchResponse">
6313         <xsd:sequence>
6314             <xsd:element name="identifier" type="xsd:anyURI"/>
6315             <xsd:choice>
6316                 <xsd:element ref="illegalState"/>
6317                 <xsd:element ref="illegalArgument"/>
6318                 <xsd:element ref="illegalAccess"/>
6319                 <xsd:element ref="illegalOperation"/>
6320                 <xsd:element ref="recipientNotAllowed"/>
6321                 <xsd:any namespace="##other" processContents="lax"/>
6322             </xsd:choice>
6323         </xsd:sequence>
6324     </xsd:complexType>
6325
6326 </xsd:schema>
6327 </wsdl:types>
6328
6329 <!-- Declaration of messages -->

```

```
6330 <wsdl:message name="addAttachment">
6331   <wsdl:part name="addAttachment" element="addAttachment"/>
6332 </wsdl:message>
6333 <wsdl:message name="addAttachmentResponse">
6334   <wsdl:part name="addAttachmentResponse" element="addAttachmentResponse"/>
6335 </wsdl:message>
6336
6337 <wsdl:message name="addComment">
6338   <wsdl:part name="addComment" element="addComment"/>
6339 </wsdl:message>
6340 <wsdl:message name="addCommentResponse">
6341   <wsdl:part name="addCommentResponse" element="addCommentResponse"/>
6342 </wsdl:message>
6343
6344 <wsdl:message name="claim">
6345   <wsdl:part name="claim" element="claim"/>
6346 </wsdl:message>
6347 <wsdl:message name="claimResponse">
6348   <wsdl:part name="claimResponse" element="claimResponse"/>
6349 </wsdl:message>
6350
6351 <wsdl:message name="batchClaim">
6352   <wsdl:part name="batchClaim" element="batchClaim"/>
6353 </wsdl:message>
6354 <wsdl:message name="batchClaimResponse">
6355   <wsdl:part name="batchClaimResponse" element="batchClaimResponse"/>
6356 </wsdl:message>
6357
6358 <wsdl:message name="complete">
6359   <wsdl:part name="complete" element="complete"/>
6360 </wsdl:message>
6361 <wsdl:message name="completeResponse">
6362   <wsdl:part name="completeResponse" element="completeResponse"/>
6363 </wsdl:message>
6364
6365 <wsdl:message name="batchComplete">
6366   <wsdl:part name="batchComplete" element="batchComplete"/>
6367 </wsdl:message>
6368 <wsdl:message name="batchCompleteResponse">
6369   <wsdl:part name="batchCompleteResponse" element="batchCompleteResponse"/>
6370 </wsdl:message>
6371
6372 <wsdl:message name="delegate">
6373   <wsdl:part name="delegate" element="delegate"/>
6374 </wsdl:message>
6375 <wsdl:message name="delegateResponse">
6376   <wsdl:part name="delegateResponse" element="delegateResponse"/>
6377 </wsdl:message>
6378
6379 <wsdl:message name="batchDelegate">
6380   <wsdl:part name="batchDelegate" element="batchDelegate"/>
6381 </wsdl:message>
6382 <wsdl:message name="batchDelegateResponse">
6383   <wsdl:part name="batchDelegateResponse" element="batchDelegateResponse"/>
6384 </wsdl:message>
6385
6386 <wsdl:message name="deleteAttachment">
6387   <wsdl:part name="deleteAttachment" element="deleteAttachment"/>
```

```

6388     </wsdl:message>
6389     <wsdl:message name="deleteAttachmentResponse">
6390         <wsdl:part name="deleteAttachmentResponse"
6391 element="deleteAttachmentResponse"/>
6392     </wsdl:message>
6393
6394     <wsdl:message name="deleteComment">
6395         <wsdl:part name="deleteComment" element="deleteComment"/>
6396     </wsdl:message>
6397     <wsdl:message name="deleteCommentResponse">
6398         <wsdl:part name="deleteCommentResponse" element="deleteCommentResponse"/>
6399     </wsdl:message>
6400
6401     <wsdl:message name="deleteFault">
6402         <wsdl:part name="deleteFault" element="deleteFault"/>
6403     </wsdl:message>
6404     <wsdl:message name="deleteFaultResponse">
6405         <wsdl:part name="deleteFaultResponse" element="deleteFaultResponse"/>
6406     </wsdl:message>
6407
6408     <wsdl:message name="deleteOutput">
6409         <wsdl:part name="deleteOutput" element="deleteOutput"/>
6410     </wsdl:message>
6411     <wsdl:message name="deleteOutputResponse">
6412         <wsdl:part name="deleteOutputResponse" element="deleteOutputResponse"/>
6413     </wsdl:message>
6414
6415     <wsdl:message name="fail">
6416         <wsdl:part name="fail" element="fail"/>
6417     </wsdl:message>
6418     <wsdl:message name="failResponse">
6419         <wsdl:part name="failResponse" element="failResponse"/>
6420     </wsdl:message>
6421
6422     <wsdl:message name="batchFail">
6423         <wsdl:part name="batchFail" element="batchFail"/>
6424     </wsdl:message>
6425     <wsdl:message name="batchFailResponse">
6426         <wsdl:part name="batchFailResponse" element="batchFailResponse"/>
6427     </wsdl:message>
6428
6429     <wsdl:message name="forward">
6430         <wsdl:part name="forward" element="forward"/>
6431     </wsdl:message>
6432     <wsdl:message name="forwardResponse">
6433         <wsdl:part name="forwardResponse" element="forwardResponse"/>
6434     </wsdl:message>
6435
6436     <wsdl:message name="batchForward">
6437         <wsdl:part name="batchForward" element="batchForward"/>
6438     </wsdl:message>
6439     <wsdl:message name="batchForwardResponse">
6440         <wsdl:part name="batchForwardResponse" element="batchForwardResponse"/>
6441     </wsdl:message>
6442
6443     <wsdl:message name="getAttachment">
6444         <wsdl:part name="getAttachment" element="getAttachment"/>
6445     </wsdl:message>

```

```

6446 <wsdl:message name="getAttachmentResponse">
6447   <wsdl:part name="getAttachmentResponse" element="getAttachmentResponse"/>
6448 </wsdl:message>
6449
6450 <wsdl:message name="getAttachmentInfos">
6451   <wsdl:part name="getAttachmentInfos" element="getAttachmentInfos"/>
6452 </wsdl:message>
6453 <wsdl:message name="getAttachmentInfosResponse">
6454   <wsdl:part name="getAttachmentInfosResponse"
6455 element="getAttachmentInfosResponse"/>
6456 </wsdl:message>
6457
6458 <wsdl:message name="getComments">
6459   <wsdl:part name="getComments" element="getComments"/>
6460 </wsdl:message>
6461 <wsdl:message name="getCommentsResponse">
6462   <wsdl:part name="getCommentsResponse" element="getCommentsResponse"/>
6463 </wsdl:message>
6464
6465 <wsdl:message name="getFault">
6466   <wsdl:part name="getFault" element="getFault"/>
6467 </wsdl:message>
6468 <wsdl:message name="getFaultResponse">
6469   <wsdl:part name="getFaultResponse" element="getFaultResponse"/>
6470 </wsdl:message>
6471
6472 <wsdl:message name="getInput">
6473   <wsdl:part name="getInput" element="getInput"/>
6474 </wsdl:message>
6475 <wsdl:message name="getInputResponse">
6476   <wsdl:part name="getInputResponse" element="getInputResponse"/>
6477 </wsdl:message>
6478
6479 <wsdl:message name="getOutcome">
6480   <wsdl:part name="getOutcome" element="getOutcome"/>
6481 </wsdl:message>
6482 <wsdl:message name="getOutcomeResponse">
6483   <wsdl:part name="getOutcomeResponse" element="getOutcomeResponse"/>
6484 </wsdl:message>
6485
6486 <wsdl:message name="getOutput">
6487   <wsdl:part name="getOutput" element="getOutput"/>
6488 </wsdl:message>
6489 <wsdl:message name="getOutputResponse">
6490   <wsdl:part name="getOutputResponse" element="getOutputResponse"/>
6491 </wsdl:message>
6492
6493 <wsdl:message name="getParentTask">
6494   <wsdl:part name="getParentTask" element="getParentTask"/>
6495 </wsdl:message>
6496 <wsdl:message name="getParentTaskResponse">
6497   <wsdl:part name="getParentTaskResponse" element="getParentTaskResponse"/>
6498 </wsdl:message>
6499
6500 <wsdl:message name="getParentTaskIdentifier">
6501   <wsdl:part name="getParentTaskIdentifier"
6502 element="getParentTaskIdentifier"/>
6503 </wsdl:message>

```

```

6504     <wsdl:message name="getParentTaskIdentifierResponse">
6505         <wsdl:part name="getParentTaskIdentifierResponse"
6506 element="getParentTaskIdentifierResponse"/>
6507     </wsdl:message>
6508
6509     <wsdl:message name="getRendering">
6510         <wsdl:part name="getRendering" element="getRendering"/>
6511     </wsdl:message>
6512     <wsdl:message name="getRenderingResponse">
6513         <wsdl:part name="getRenderingResponse" element="getRenderingResponse"/>
6514     </wsdl:message>
6515
6516     <wsdl:message name="getRenderingTypes">
6517         <wsdl:part name="getRenderingTypes" element="getRenderingTypes"/>
6518     </wsdl:message>
6519     <wsdl:message name="getRenderingTypesResponse">
6520         <wsdl:part name="getRenderingTypesResponse"
6521 element="getRenderingTypesResponse"/>
6522     </wsdl:message>
6523
6524     <wsdl:message name="getSubtaskIdentifiers">
6525         <wsdl:part name="getSubtaskIdentifiers" element="getSubtaskIdentifiers"/>
6526     </wsdl:message>
6527     <wsdl:message name="getSubtaskIdentifiersResponse">
6528         <wsdl:part name="getSubtaskIdentifiersResponse"
6529 element="getSubtaskIdentifiersResponse"/>
6530     </wsdl:message>
6531
6532     <wsdl:message name="getSubtasks">
6533         <wsdl:part name="getSubtasks" element="getSubtasks"/>
6534     </wsdl:message>
6535     <wsdl:message name="getSubtasksResponse">
6536         <wsdl:part name="getSubtasksResponse" element="getSubtasksResponse"/>
6537     </wsdl:message>
6538
6539     <wsdl:message name="getTaskDescription">
6540         <wsdl:part name="getTaskDescription" element="getTaskDescription"/>
6541     </wsdl:message>
6542     <wsdl:message name="getTaskDescriptionResponse">
6543         <wsdl:part name="getTaskDescriptionResponse"
6544 element="getTaskDescriptionResponse"/>
6545     </wsdl:message>
6546
6547     <wsdl:message name="getTaskDetails">
6548         <wsdl:part name="getTaskDetails" element="getTaskDetails"/>
6549     </wsdl:message>
6550     <wsdl:message name="getTaskDetailsResponse">
6551         <wsdl:part name="getTaskDetailsResponse"
6552 element="getTaskDetailsResponse"/>
6553     </wsdl:message>
6554
6555     <wsdl:message name="getTaskHistory">
6556         <wsdl:part name="getTaskHistory" element="getTaskHistory"/>
6557     </wsdl:message>
6558     <wsdl:message name="getTaskHistoryResponse">
6559         <wsdl:part name="getTaskHistoryResponse"
6560 element="getTaskHistoryResponse"/>
6561     </wsdl:message>

```

```

6562
6563     <wsdl:message name="getTaskInstanceData">
6564         <wsdl:part name="getTaskInstanceData" element="getTaskInstanceData"/>
6565     </wsdl:message>
6566     <wsdl:message name="getTaskInstanceDataResponse">
6567         <wsdl:part name="getTaskInstanceDataResponse"
6568 element="getTaskInstanceDataResponse"/>
6569     </wsdl:message>
6570
6571     <wsdl:message name="getTaskOperations">
6572         <wsdl:part name="getTaskOperations" element="getTaskOperations"/>
6573     </wsdl:message>
6574     <wsdl:message name="getTaskOperationsResponse">
6575         <wsdl:part name="getTaskOperationsResponse"
6576 element="getTaskOperationsResponse"/>
6577     </wsdl:message>
6578
6579     <wsdl:message name="hasSubtasks">
6580         <wsdl:part name="hasSubtasks" element="hasSubtasks"/>
6581     </wsdl:message>
6582     <wsdl:message name="hasSubtasksResponse">
6583         <wsdl:part name="hasSubtasksResponse" element="hasSubtasksResponse"/>
6584     </wsdl:message>
6585
6586     <wsdl:message name="instantiateSubtask">
6587         <wsdl:part name="instantiateSubtask" element="instantiateSubtask"/>
6588     </wsdl:message>
6589     <wsdl:message name="instantiateSubtaskResponse">
6590         <wsdl:part name="instantiateSubtaskResponse"
6591 element="instantiateSubtaskResponse"/>
6592     </wsdl:message>
6593
6594     <wsdl:message name="isSubtask">
6595         <wsdl:part name="isSubtask" element="isSubtask"/>
6596     </wsdl:message>
6597     <wsdl:message name="isSubtaskResponse">
6598         <wsdl:part name="isSubtaskResponse" element="isSubtaskResponse"/>
6599     </wsdl:message>
6600
6601     <wsdl:message name="release">
6602         <wsdl:part name="release" element="release"/>
6603     </wsdl:message>
6604     <wsdl:message name="releaseResponse">
6605         <wsdl:part name="releaseResponse" element="releaseResponse"/>
6606     </wsdl:message>
6607
6608     <wsdl:message name="batchRelease">
6609         <wsdl:part name="batchRelease" element="batchRelease"/>
6610     </wsdl:message>
6611     <wsdl:message name="batchReleaseResponse">
6612         <wsdl:part name="batchReleaseResponse" element="batchReleaseResponse"/>
6613     </wsdl:message>
6614
6615     <wsdl:message name="remove">
6616         <wsdl:part name="remove" element="remove"/>
6617     </wsdl:message>
6618     <wsdl:message name="removeResponse">
6619         <wsdl:part name="removeResponse" element="removeResponse"/>

```



```

6620 </wsdl:message>
6621
6622 <wsdl:message name="batchRemove">
6623   <wsdl:part name="batchRemove" element="batchRemove"/>
6624 </wsdl:message>
6625 <wsdl:message name="batchRemoveResponse">
6626   <wsdl:part name="batchRemoveResponse" element="batchRemoveResponse"/>
6627 </wsdl:message>
6628
6629 <wsdl:message name="resume">
6630   <wsdl:part name="resume" element="resume"/>
6631 </wsdl:message>
6632 <wsdl:message name="resumeResponse">
6633   <wsdl:part name="resumeResponse" element="resumeResponse"/>
6634 </wsdl:message>
6635
6636 <wsdl:message name="batchResume">
6637   <wsdl:part name="batchResume" element="batchResume"/>
6638 </wsdl:message>
6639 <wsdl:message name="batchResumeResponse">
6640   <wsdl:part name="batchResumeResponse" element="batchResumeResponse"/>
6641 </wsdl:message>
6642
6643 <wsdl:message name="setFault">
6644   <wsdl:part name="setFault" element="setFault"/>
6645 </wsdl:message>
6646 <wsdl:message name="setFaultResponse">
6647   <wsdl:part name="setFaultResponse" element="setFaultResponse"/>
6648 </wsdl:message>
6649
6650 <wsdl:message name="setOutput">
6651   <wsdl:part name="setOutput" element="setOutput"/>
6652 </wsdl:message>
6653 <wsdl:message name="setOutputResponse">
6654   <wsdl:part name="setOutputResponse" element="setOutputResponse"/>
6655 </wsdl:message>
6656
6657 <wsdl:message name="setPriority">
6658   <wsdl:part name="setPriority" element="setPriority"/>
6659 </wsdl:message>
6660 <wsdl:message name="setPriorityResponse">
6661   <wsdl:part name="setPriorityResponse" element="setPriorityResponse"/>
6662 </wsdl:message>
6663
6664 <wsdl:message name="batchSetPriority">
6665   <wsdl:part name="batchSetPriority" element="batchSetPriority"/>
6666 </wsdl:message>
6667 <wsdl:message name="batchSetPriorityResponse">
6668   <wsdl:part name="batchSetPriorityResponse"
6669 element="batchSetPriorityResponse"/>
6670 </wsdl:message>
6671
6672 <wsdl:message name="setTaskCompletionDeadlineExpression">
6673   <wsdl:part name="setTaskCompletionDeadlineExpression"
6674 element="setTaskCompletionDeadlineExpression"/>
6675 </wsdl:message>
6676 <wsdl:message name="setTaskCompletionDeadlineExpressionResponse">

```



```

6677     <wsdl:part name="setTaskCompletionDeadlineExpressionResponse"
6678 element="setTaskCompletionDeadlineExpressionResponse"/>
6679   </wsdl:message>
6680
6681   <wsdl:message name="setTaskCompletionDurationExpression">
6682     <wsdl:part name="setTaskCompletionDurationExpression"
6683 element="setTaskCompletionDurationExpression"/>
6684   </wsdl:message>
6685   <wsdl:message name="setTaskCompletionDurationExpressionResponse">
6686     <wsdl:part name="setTaskCompletionDurationExpressionResponse"
6687 element="setTaskCompletionDurationExpressionResponse"/>
6688   </wsdl:message>
6689
6690   <wsdl:message name="setTaskStartDeadlineExpression">
6691     <wsdl:part name="setTaskStartDeadlineExpression"
6692 element="setTaskStartDeadlineExpression"/>
6693   </wsdl:message>
6694   <wsdl:message name="setTaskStartDeadlineExpressionResponse">
6695     <wsdl:part name="setTaskStartDeadlineExpressionResponse"
6696 element="setTaskStartDeadlineExpressionResponse"/>
6697   </wsdl:message>
6698
6699   <wsdl:message name="setTaskStartDurationExpression">
6700     <wsdl:part name="setTaskStartDurationExpression"
6701 element="setTaskStartDurationExpression"/>
6702   </wsdl:message>
6703   <wsdl:message name="setTaskStartDurationExpressionResponse">
6704     <wsdl:part name="setTaskStartDurationExpressionResponse"
6705 element="setTaskStartDurationExpressionResponse"/>
6706   </wsdl:message>
6707
6708   <wsdl:message name="skip">
6709     <wsdl:part name="skip" element="skip"/>
6710   </wsdl:message>
6711   <wsdl:message name="skipResponse">
6712     <wsdl:part name="skipResponse" element="skipResponse"/>
6713   </wsdl:message>
6714
6715   <wsdl:message name="batchSkip">
6716     <wsdl:part name="batchSkip" element="batchSkip"/>
6717   </wsdl:message>
6718   <wsdl:message name="batchSkipResponse">
6719     <wsdl:part name="batchSkipResponse" element="batchSkipResponse"/>
6720   </wsdl:message>
6721
6722   <wsdl:message name="start">
6723     <wsdl:part name="start" element="start"/>
6724   </wsdl:message>
6725   <wsdl:message name="startResponse">
6726     <wsdl:part name="startResponse" element="startResponse"/>
6727   </wsdl:message>
6728
6729   <wsdl:message name="batchStart">
6730     <wsdl:part name="batchStart" element="batchStart"/>
6731   </wsdl:message>
6732   <wsdl:message name="batchStartResponse">
6733     <wsdl:part name="batchStartResponse" element="batchStartResponse"/>
6734   </wsdl:message>

```

```

6735
6736 <wsdl:message name="stop">
6737   <wsdl:part name="stop" element="stop"/>
6738 </wsdl:message>
6739 <wsdl:message name="stopResponse">
6740   <wsdl:part name="stopResponse" element="stopResponse"/>
6741 </wsdl:message>
6742
6743 <wsdl:message name="batchStop">
6744   <wsdl:part name="batchStop" element="batchStop"/>
6745 </wsdl:message>
6746 <wsdl:message name="batchStopResponse">
6747   <wsdl:part name="batchStopResponse" element="batchStopResponse"/>
6748 </wsdl:message>
6749
6750 <wsdl:message name="suspend">
6751   <wsdl:part name="suspend" element="suspend"/>
6752 </wsdl:message>
6753 <wsdl:message name="suspendResponse">
6754   <wsdl:part name="suspendResponse" element="suspendResponse"/>
6755 </wsdl:message>
6756
6757 <wsdl:message name="batchSuspend">
6758   <wsdl:part name="batchSuspend" element="batchSuspend"/>
6759 </wsdl:message>
6760 <wsdl:message name="batchSuspendResponse">
6761   <wsdl:part name="batchSuspendResponse" element="batchSuspendResponse"/>
6762 </wsdl:message>
6763
6764 <wsdl:message name="suspendUntil">
6765   <wsdl:part name="suspendUntil" element="suspendUntil"/>
6766 </wsdl:message>
6767 <wsdl:message name="suspendUntilResponse">
6768   <wsdl:part name="suspendUntilResponse" element="suspendUntilResponse"/>
6769 </wsdl:message>
6770
6771 <wsdl:message name="batchSuspendUntil">
6772   <wsdl:part name="batchSuspendUntil" element="batchSuspendUntil"/>
6773 </wsdl:message>
6774 <wsdl:message name="batchSuspendUntilResponse">
6775   <wsdl:part name="batchSuspendUntilResponse"
6776 element="batchSuspendUntilResponse"/>
6777 </wsdl:message>
6778
6779 <wsdl:message name="updateComment">
6780   <wsdl:part name="updateComment" element="updateComment"/>
6781 </wsdl:message>
6782 <wsdl:message name="updateCommentResponse">
6783   <wsdl:part name="updateCommentResponse" element="updateCommentResponse"/>
6784 </wsdl:message>
6785
6786 <wsdl:message name="getMyTaskAbstracts">
6787   <wsdl:part name="getMyTaskAbstracts" element="getMyTaskAbstracts"/>
6788 </wsdl:message>
6789 <wsdl:message name="getMyTaskAbstractsResponse">
6790   <wsdl:part name="getMyTaskAbstractsResponse"
6791 element="getMyTaskAbstractsResponse"/>
6792 </wsdl:message>

```

```

6793 <wsdl:message name="getMyTaskDetails">
6794   <wsdl:part name="getMyTaskDetails" element="getMyTaskDetails"/>
6795 </wsdl:message>
6797 <wsdl:message name="getMyTaskDetailsResponse">
6798   <wsdl:part name="getMyTaskDetailsResponse"
6799 element="getMyTaskDetailsResponse"/>
6800 </wsdl:message>
6801
6802 <wsdl:message name="query">
6803   <wsdl:part name="query" element="query"/>
6804 </wsdl:message>
6805 <wsdl:message name="queryResponse">
6806   <wsdl:part name="queryResponse" element="queryResponse"/>
6807 </wsdl:message>
6808
6809 <wsdl:message name="activate">
6810   <wsdl:part name="activate" element="activate"/>
6811 </wsdl:message>
6812 <wsdl:message name="activateResponse">
6813   <wsdl:part name="activateResponse" element="activateResponse"/>
6814 </wsdl:message>
6815
6816 <wsdl:message name="batchActivate">
6817   <wsdl:part name="batchActivate" element="batchActivate"/>
6818 </wsdl:message>
6819 <wsdl:message name="batchActivateResponse">
6820   <wsdl:part name="batchActivateResponse" element="batchActivateResponse"/>
6821 </wsdl:message>
6822
6823 <wsdl:message name="nominate">
6824   <wsdl:part name="nominate" element="nominate"/>
6825 </wsdl:message>
6826 <wsdl:message name="nominateResponse">
6827   <wsdl:part name="nominateResponse" element="nominateResponse"/>
6828 </wsdl:message>
6829
6830 <wsdl:message name="batchNominate">
6831   <wsdl:part name="batchNominate" element="batchNominate"/>
6832 </wsdl:message>
6833 <wsdl:message name="batchNominateResponse">
6834   <wsdl:part name="batchNominateResponse" element="batchNominateResponse"/>
6835 </wsdl:message>
6836
6837 <wsdl:message name="setGenericHumanRole">
6838   <wsdl:part name="setGenericHumanRole" element="setGenericHumanRole"/>
6839 </wsdl:message>
6840 <wsdl:message name="setGenericHumanRoleResponse">
6841   <wsdl:part name="setGenericHumanRoleResponse"
6842 element="setGenericHumanRoleResponse"/>
6843 </wsdl:message>
6844
6845 <wsdl:message name="batchSetGenericHumanRole">
6846   <wsdl:part name="batchSetGenericHumanRole"
6847 element="batchSetGenericHumanRole"/>
6848 </wsdl:message>
6849 <wsdl:message name="batchSetGenericHumanRoleResponse">

```

```

6850     <wsdl:part name="batchSetGenericHumanRoleResponse"
6851 element="batchSetGenericHumanRoleResponse"/>
6852 </wsdl:message>
6853
6854 <!-- Declaration of fault messages -->
6855 <wsdl:message name="illegalStateFault">
6856   <wsdl:part name="illegalState" element="illegalState"/>
6857 </wsdl:message>
6858 <wsdl:message name="illegalArgumentFault">
6859   <wsdl:part name="illegalArgument" element="illegalArgument"/>
6860 </wsdl:message>
6861 <wsdl:message name="illegalAccessFault">
6862   <wsdl:part name="illegalAccess" element="illegalAccess"/>
6863 </wsdl:message>
6864 <wsdl:message name="illegalOperationFault">
6865   <wsdl:part name="illegalOperation" element="illegalOperation"/>
6866 </wsdl:message>
6867 <wsdl:message name="recipientNotAllowed">
6868   <wsdl:part name="recipientNotAllowed" element="recipientNotAllowed"/>
6869 </wsdl:message>
6870
6871 <!-- Port type definition -->
6872 <wsdl:portType name="taskOperations">
6873
6874   <wsdl:operation name="addAttachment">
6875     <wsdl:input message="addAttachment"/>
6876     <wsdl:output message="addAttachmentResponse"/>
6877     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
6878     <wsdl:fault name="illegalArgumentFault"
6879 message="illegalArgumentFault"/>
6880     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
6881     <wsdl:fault name="illegalOperationFault"
6882 message="illegalOperationFault"/>
6883   </wsdl:operation>
6884
6885   <wsdl:operation name="addComment">
6886     <wsdl:input message="addComment"/>
6887     <wsdl:output message="addCommentResponse"/>
6888     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
6889     <wsdl:fault name="illegalArgumentFault"
6890 message="illegalArgumentFault"/>
6891     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
6892     <wsdl:fault name="illegalOperationFault"
6893 message="illegalOperationFault"/>
6894   </wsdl:operation>
6895
6896   <wsdl:operation name="claim">
6897     <wsdl:input message="claim"/>
6898     <wsdl:output message="claimResponse"/>
6899     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
6900     <wsdl:fault name="illegalArgumentFault"
6901 message="illegalArgumentFault"/>
6902     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
6903     <wsdl:fault name="illegalOperationFault"
6904 message="illegalOperationFault"/>
6905   </wsdl:operation>
6906
6907   <wsdl:operation name="batchClaim">

```

```

6908     <wsdl:input message="batchClaim"/>
6909     <wsdl:output message="batchClaimResponse"/>
6910 </wsdl:operation>
6911
6912 <wsdl:operation name="complete">
6913     <wsdl:input message="complete"/>
6914     <wsdl:output message="completeResponse"/>
6915     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
6916     <wsdl:fault name="illegalArgumentFault"
6917 message="illegalArgumentFault"/>
6918     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
6919     <wsdl:fault name="illegalOperationFault"
6920 message="illegalOperationFault"/>
6921 </wsdl:operation>
6922
6923 <wsdl:operation name="batchComplete">
6924     <wsdl:input message="batchComplete"/>
6925     <wsdl:output message="batchCompleteResponse"/>
6926 </wsdl:operation>
6927
6928 <wsdl:operation name="delegate">
6929     <wsdl:input message="delegate"/>
6930     <wsdl:output message="delegateResponse"/>
6931     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
6932     <wsdl:fault name="illegalArgumentFault"
6933 message="illegalArgumentFault"/>
6934     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
6935     <wsdl:fault name="illegalOperationFault"
6936 message="illegalOperationFault"/>
6937     <wsdl:fault name="recipientNotAllowed" message="recipientNotAllowed"/>
6938 </wsdl:operation>
6939
6940 <wsdl:operation name="batchDelegate">
6941     <wsdl:input message="batchDelegate"/>
6942     <wsdl:output message="batchDelegateResponse"/>
6943 </wsdl:operation>
6944
6945 <wsdl:operation name="deleteAttachment">
6946     <wsdl:input message="deleteAttachment"/>
6947     <wsdl:output message="deleteAttachmentResponse"/>
6948     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
6949     <wsdl:fault name="illegalArgumentFault"
6950 message="illegalArgumentFault"/>
6951     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
6952     <wsdl:fault name="illegalOperationFault"
6953 message="illegalOperationFault"/>
6954 </wsdl:operation>
6955
6956 <wsdl:operation name="deleteComment">
6957     <wsdl:input message="deleteComment"/>
6958     <wsdl:output message="deleteCommentResponse"/>
6959     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
6960     <wsdl:fault name="illegalArgumentFault"
6961 message="illegalArgumentFault"/>
6962     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
6963     <wsdl:fault name="illegalOperationFault"
6964 message="illegalOperationFault"/>
6965 </wsdl:operation>

```

```

6966
6967     <wsdl:operation name="deleteFault">
6968         <wsdl:input message="deleteFault"/>
6969         <wsdl:output message="deleteFaultResponse"/>
6970         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
6971         <wsdl:fault name="illegalArgumentFault"
6972 message="illegalArgumentFault"/>
6973         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
6974         <wsdl:fault name="illegalOperationFault"
6975 message="illegalOperationFault"/>
6976     </wsdl:operation>
6977
6978     <wsdl:operation name="deleteOutput">
6979         <wsdl:input message="deleteOutput"/>
6980         <wsdl:output message="deleteOutputResponse"/>
6981         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
6982         <wsdl:fault name="illegalArgumentFault"
6983 message="illegalArgumentFault"/>
6984         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
6985         <wsdl:fault name="illegalOperationFault"
6986 message="illegalOperationFault"/>
6987     </wsdl:operation>
6988
6989     <wsdl:operation name="fail">
6990         <wsdl:input message="fail"/>
6991         <wsdl:output message="failResponse"/>
6992         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
6993         <wsdl:fault name="illegalArgumentFault"
6994 message="illegalArgumentFault"/>
6995         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
6996         <wsdl:fault name="illegalOperationFault"
6997 message="illegalOperationFault"/>
6998     </wsdl:operation>
6999
7000     <wsdl:operation name="batchFail">
7001         <wsdl:input message="batchFail"/>
7002         <wsdl:output message="batchFailResponse"/>
7003     </wsdl:operation>
7004
7005     <wsdl:operation name="forward">
7006         <wsdl:input message="forward"/>
7007         <wsdl:output message="forwardResponse"/>
7008         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7009         <wsdl:fault name="illegalArgumentFault"
7010 message="illegalArgumentFault"/>
7011         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7012         <wsdl:fault name="illegalOperationFault"
7013 message="illegalOperationFault"/>
7014     </wsdl:operation>
7015
7016     <wsdl:operation name="batchForward">
7017         <wsdl:input message="batchForward"/>
7018         <wsdl:output message="batchForwardResponse"/>
7019     </wsdl:operation>
7020
7021     <wsdl:operation name="getAttachment">
7022         <wsdl:input message="getAttachment"/>
7023         <wsdl:output message="getAttachmentResponse"/>

```



```

7024     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7025     <wsdl:fault name="illegalArgumentFault"
7026 message="illegalArgumentFault"/>
7027     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7028     <wsdl:fault name="illegalOperationFault"
7029 message="illegalOperationFault"/>
7030 </wsdl:operation>
7031
7032     <wsdl:operation name="getAttachmentInfos">
7033     <wsdl:input message="getAttachmentInfos"/>
7034     <wsdl:output message="getAttachmentInfosResponse"/>
7035     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7036     <wsdl:fault name="illegalArgumentFault"
7037 message="illegalArgumentFault"/>
7038     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7039     <wsdl:fault name="illegalOperationFault"
7040 message="illegalOperationFault"/>
7041 </wsdl:operation>
7042
7043     <wsdl:operation name="getComments">
7044     <wsdl:input message="getComments"/>
7045     <wsdl:output message="getCommentsResponse"/>
7046     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7047     <wsdl:fault name="illegalArgumentFault"
7048 message="illegalArgumentFault"/>
7049     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7050     <wsdl:fault name="illegalOperationFault"
7051 message="illegalOperationFault"/>
7052 </wsdl:operation>
7053
7054     <wsdl:operation name="getFault">
7055     <wsdl:input message="getFault"/>
7056     <wsdl:output message="getFaultResponse"/>
7057     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7058     <wsdl:fault name="illegalArgumentFault"
7059 message="illegalArgumentFault"/>
7060     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7061     <wsdl:fault name="illegalOperationFault"
7062 message="illegalOperationFault"/>
7063 </wsdl:operation>
7064
7065     <wsdl:operation name="getInput">
7066     <wsdl:input message="getInput"/>
7067     <wsdl:output message="getInputResponse"/>
7068     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7069     <wsdl:fault name="illegalArgumentFault"
7070 message="illegalArgumentFault"/>
7071     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7072     <wsdl:fault name="illegalOperationFault"
7073 message="illegalOperationFault"/>
7074 </wsdl:operation>
7075
7076     <wsdl:operation name="getOutcome">
7077     <wsdl:input message="getOutcome"/>
7078     <wsdl:output message="getOutcomeResponse"/>
7079     <wsdl:fault name="illegalArgumentFault"
7080 message="illegalArgumentFault"/>

```

```

7081     <wsdl:fault name="illegalOperationFault"
7082 message="illegalOperationFault"/>
7083   </wsdl:operation>
7084
7085   <wsdl:operation name="getOutput">
7086     <wsdl:input message="getOutput"/>
7087     <wsdl:output message="getOutputResponse"/>
7088     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7089     <wsdl:fault name="illegalArgumentFault"
7090 message="illegalArgumentFault"/>
7091     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7092     <wsdl:fault name="illegalOperationFault"
7093 message="illegalOperationFault"/>
7094   </wsdl:operation>
7095
7096   <wsdl:operation name="getParentTask">
7097     <wsdl:input message="getParentTask"/>
7098     <wsdl:output message="getParentTaskResponse"/>
7099     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7100     <wsdl:fault name="illegalArgumentFault"
7101 message="illegalArgumentFault"/>
7102     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7103     <wsdl:fault name="illegalOperationFault"
7104 message="illegalOperationFault"/>
7105   </wsdl:operation>
7106
7107   <wsdl:operation name="getParentTaskIdentifier">
7108     <wsdl:input message="getParentTaskIdentifier"/>
7109     <wsdl:output message="getParentTaskIdentifierResponse"/>
7110     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7111     <wsdl:fault name="illegalArgumentFault"
7112 message="illegalArgumentFault"/>
7113     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7114     <wsdl:fault name="illegalOperationFault"
7115 message="illegalOperationFault"/>
7116   </wsdl:operation>
7117
7118   <wsdl:operation name="getRendering">
7119     <wsdl:input message="getRendering"/>
7120     <wsdl:output message="getRenderingResponse"/>
7121     <wsdl:fault name="illegalArgumentFault"
7122 message="illegalArgumentFault"/>
7123   </wsdl:operation>
7124
7125   <wsdl:operation name="getRenderingTypes">
7126     <wsdl:input message="getRenderingTypes"/>
7127     <wsdl:output message="getRenderingTypesResponse"/>
7128     <wsdl:fault name="illegalArgumentFault"
7129 message="illegalArgumentFault"/>
7130   </wsdl:operation>
7131
7132   <wsdl:operation name="getSubtaskIdentifiers">
7133     <wsdl:input message="getSubtaskIdentifiers"/>
7134     <wsdl:output message="getSubtaskIdentifiersResponse"/>
7135     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7136     <wsdl:fault name="illegalArgumentFault"
7137 message="illegalArgumentFault"/>
7138     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>

```



```

7139     <wsdl:fault name="illegalOperationFault"
7140 message="illegalOperationFault"/>
7141   </wsdl:operation>
7142
7143   <wsdl:operation name="getSubtasks">
7144     <wsdl:input message="getSubtasks"/>
7145     <wsdl:output message="getSubtasksResponse"/>
7146     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7147     <wsdl:fault name="illegalArgumentFault"
7148 message="illegalArgumentFault"/>
7149     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7150     <wsdl:fault name="illegalOperationFault"
7151 message="illegalOperationFault"/>
7152   </wsdl:operation>
7153
7154   <wsdl:operation name="getTaskDescription">
7155     <wsdl:input message="getTaskDescription"/>
7156     <wsdl:output message="getTaskDescriptionResponse"/>
7157     <wsdl:fault name="illegalArgumentFault"
7158 message="illegalArgumentFault"/>
7159   </wsdl:operation>
7160
7161   <wsdl:operation name="getTaskDetails">
7162     <wsdl:input message="getTaskDetails"/>
7163     <wsdl:output message="getTaskDetailsResponse"/>
7164     <wsdl:fault name="illegalArgumentFault"
7165 message="illegalArgumentFault"/>
7166   </wsdl:operation>
7167
7168   <wsdl:operation name="getTaskHistory">
7169     <wsdl:input message="getTaskHistory"/>
7170     <wsdl:output message="getTaskHistoryResponse"/>
7171     <wsdl:fault name="illegalArgumentFault"
7172 message="illegalArgumentFault"/>
7173     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7174     <wsdl:fault name="illegalOperationFault"
7175 message="illegalOperationFault"/>
7176   </wsdl:operation>
7177
7178   <wsdl:operation name="getTaskInstanceData">
7179     <wsdl:input message="getTaskInstanceData"/>
7180     <wsdl:output message="getTaskInstanceDataResponse"/>
7181     <wsdl:fault name="illegalArgumentFault"
7182 message="illegalArgumentFault"/>
7183     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7184     <wsdl:fault name="illegalOperationFault"
7185 message="illegalOperationFault"/>
7186   </wsdl:operation>
7187
7188   <wsdl:operation name="getTaskOperations">
7189     <wsdl:input message="getTaskOperations"/>
7190     <wsdl:output message="getTaskOperationsResponse"/>
7191     <wsdl:fault name="illegalArgumentFault"
7192 message="illegalArgumentFault"/>
7193     <wsdl:fault name="illegalOperationFault"
7194 message="illegalOperationFault"/>
7195   </wsdl:operation>
7196

```

```

7197     <wsdl:operation name="hasSubtasks">
7198         <wsdl:input message="hasSubtasks"/>
7199         <wsdl:output message="hasSubtasksResponse"/>
7200         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7201         <wsdl:fault name="illegalArgumentFault"
7202 message="illegalArgumentFault"/>
7203         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7204         <wsdl:fault name="illegalOperationFault"
7205 message="illegalOperationFault"/>
7206     </wsdl:operation>
7207
7208     <wsdl:operation name="instantiateSubtask">
7209         <wsdl:input message="instantiateSubtask"/>
7210         <wsdl:output message="instantiateSubtaskResponse"/>
7211         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7212         <wsdl:fault name="illegalArgumentFault"
7213 message="illegalArgumentFault"/>
7214         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7215         <wsdl:fault name="illegalOperationFault"
7216 message="illegalOperationFault"/>
7217     </wsdl:operation>
7218
7219     <wsdl:operation name="isSubtask">
7220         <wsdl:input message="isSubtask"/>
7221         <wsdl:output message="isSubtaskResponse"/>
7222         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7223         <wsdl:fault name="illegalArgumentFault"
7224 message="illegalArgumentFault"/>
7225         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7226         <wsdl:fault name="illegalOperationFault"
7227 message="illegalOperationFault"/>
7228     </wsdl:operation>
7229
7230     <wsdl:operation name="release">
7231         <wsdl:input message="release"/>
7232         <wsdl:output message="releaseResponse"/>
7233         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7234         <wsdl:fault name="illegalArgumentFault"
7235 message="illegalArgumentFault"/>
7236         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7237         <wsdl:fault name="illegalOperationFault"
7238 message="illegalOperationFault"/>
7239     </wsdl:operation>
7240
7241     <wsdl:operation name="batchRelease">
7242         <wsdl:input message="batchRelease"/>
7243         <wsdl:output message="batchReleaseResponse"/>
7244     </wsdl:operation>
7245
7246     <wsdl:operation name="remove">
7247         <wsdl:input message="remove"/>
7248         <wsdl:output message="removeResponse"/>
7249         <wsdl:fault name="illegalArgumentFault"
7250 message="illegalArgumentFault"/>
7251         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7252         <wsdl:fault name="illegalOperationFault"
7253 message="illegalOperationFault"/>
7254     </wsdl:operation>

```

```

7255
7256     <wsdl:operation name="batchRemove">
7257         <wsdl:input message="batchRemove"/>
7258         <wsdl:output message="batchRemoveResponse"/>
7259     </wsdl:operation>
7260
7261     <wsdl:operation name="resume">
7262         <wsdl:input message="resume"/>
7263         <wsdl:output message="resumeResponse"/>
7264         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7265         <wsdl:fault name="illegalArgumentFault"
7266 message="illegalArgumentFault"/>
7267         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7268         <wsdl:fault name="illegalOperationFault"
7269 message="illegalOperationFault"/>
7270     </wsdl:operation>
7271
7272     <wsdl:operation name="batchResume">
7273         <wsdl:input message="batchResume"/>
7274         <wsdl:output message="batchResumeResponse"/>
7275     </wsdl:operation>
7276
7277     <wsdl:operation name="setFault">
7278         <wsdl:input message="setFault"/>
7279         <wsdl:output message="setFaultResponse"/>
7280         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7281         <wsdl:fault name="illegalArgumentFault"
7282 message="illegalArgumentFault"/>
7283         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7284         <wsdl:fault name="illegalOperationFault"
7285 message="illegalOperationFault"/>
7286     </wsdl:operation>
7287
7288     <wsdl:operation name="setOutput">
7289         <wsdl:input message="setOutput"/>
7290         <wsdl:output message="setOutputResponse"/>
7291         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7292         <wsdl:fault name="illegalArgumentFault"
7293 message="illegalArgumentFault"/>
7294         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7295         <wsdl:fault name="illegalOperationFault"
7296 message="illegalOperationFault"/>
7297     </wsdl:operation>
7298
7299     <wsdl:operation name="setPriority">
7300         <wsdl:input message="setPriority"/>
7301         <wsdl:output message="setPriorityResponse"/>
7302         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7303         <wsdl:fault name="illegalArgumentFault"
7304 message="illegalArgumentFault"/>
7305         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7306         <wsdl:fault name="illegalOperationFault"
7307 message="illegalOperationFault"/>
7308     </wsdl:operation>
7309
7310     <wsdl:operation name="batchSetPriority">
7311         <wsdl:input message="batchSetPriority"/>
7312         <wsdl:output message="batchSetPriorityResponse"/>

```

```

7313     </wsdl:operation>
7314
7315     <wsdl:operation name="setTaskCompletionDeadlineExpression">
7316         <wsdl:input message="setTaskCompletionDeadlineExpression"/>
7317         <wsdl:output message="setTaskCompletionDeadlineExpressionResponse"/>
7318         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7319         <wsdl:fault name="illegalArgumentFault"
7320 message="illegalArgumentFault"/>
7321         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7322         <wsdl:fault name="illegalOperationFault"
7323 message="illegalOperationFault"/>
7324     </wsdl:operation>
7325
7326     <wsdl:operation name="setTaskCompletionDurationExpression">
7327         <wsdl:input message="setTaskCompletionDurationExpression"/>
7328         <wsdl:output message="setTaskCompletionDurationExpressionResponse"/>
7329         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7330         <wsdl:fault name="illegalArgumentFault"
7331 message="illegalArgumentFault"/>
7332         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7333         <wsdl:fault name="illegalOperationFault"
7334 message="illegalOperationFault"/>
7335     </wsdl:operation>
7336
7337     <wsdl:operation name="setTaskStartDeadlineExpression">
7338         <wsdl:input message="setTaskStartDeadlineExpression"/>
7339         <wsdl:output message="setTaskStartDeadlineExpressionResponse"/>
7340         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7341         <wsdl:fault name="illegalArgumentFault"
7342 message="illegalArgumentFault"/>
7343         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7344         <wsdl:fault name="illegalOperationFault"
7345 message="illegalOperationFault"/>
7346     </wsdl:operation>
7347
7348     <wsdl:operation name="setTaskStartDurationExpression">
7349         <wsdl:input message="setTaskStartDurationExpression"/>
7350         <wsdl:output message="setTaskStartDurationExpressionResponse"/>
7351         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7352         <wsdl:fault name="illegalArgumentFault"
7353 message="illegalArgumentFault"/>
7354         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7355         <wsdl:fault name="illegalOperationFault"
7356 message="illegalOperationFault"/>
7357     </wsdl:operation>
7358
7359     <wsdl:operation name="skip">
7360         <wsdl:input message="skip"/>
7361         <wsdl:output message="skipResponse"/>
7362         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7363         <wsdl:fault name="illegalArgumentFault"
7364 message="illegalArgumentFault"/>
7365         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7366         <wsdl:fault name="illegalOperationFault"
7367 message="illegalOperationFault"/>
7368     </wsdl:operation>
7369
7370     <wsdl:operation name="batchSkip">

```

```

7371     <wsdl:input message="batchSkip"/>
7372     <wsdl:output message="batchSkipResponse"/>
7373 </wsdl:operation>
7374
7375     <wsdl:operation name="start">
7376         <wsdl:input message="start"/>
7377         <wsdl:output message="startResponse"/>
7378         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7379         <wsdl:fault name="illegalArgumentFault"
7380 message="illegalArgumentFault"/>
7381         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7382         <wsdl:fault name="illegalOperationFault"
7383 message="illegalOperationFault"/>
7384     </wsdl:operation>
7385
7386     <wsdl:operation name="batchStart">
7387         <wsdl:input message="batchStart"/>
7388         <wsdl:output message="batchStartResponse"/>
7389     </wsdl:operation>
7390
7391     <wsdl:operation name="stop">
7392         <wsdl:input message="stop"/>
7393         <wsdl:output message="stopResponse"/>
7394         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7395         <wsdl:fault name="illegalArgumentFault"
7396 message="illegalArgumentFault"/>
7397         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7398         <wsdl:fault name="illegalOperationFault"
7399 message="illegalOperationFault"/>
7400     </wsdl:operation>
7401
7402     <wsdl:operation name="batchStop">
7403         <wsdl:input message="batchStop"/>
7404         <wsdl:output message="batchStopResponse"/>
7405     </wsdl:operation>
7406
7407     <wsdl:operation name="suspend">
7408         <wsdl:input message="suspend"/>
7409         <wsdl:output message="suspendResponse"/>
7410         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7411         <wsdl:fault name="illegalArgumentFault"
7412 message="illegalArgumentFault"/>
7413         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7414         <wsdl:fault name="illegalOperationFault"
7415 message="illegalOperationFault"/>
7416     </wsdl:operation>
7417
7418     <wsdl:operation name="batchSuspend">
7419         <wsdl:input message="batchSuspend"/>
7420         <wsdl:output message="batchSuspendResponse"/>
7421     </wsdl:operation>
7422
7423     <wsdl:operation name="suspendUntil">
7424         <wsdl:input message="suspendUntil"/>
7425         <wsdl:output message="suspendUntilResponse"/>
7426         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7427         <wsdl:fault name="illegalArgumentFault"
7428 message="illegalArgumentFault"/>

```

```

7429     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7430     <wsdl:fault name="illegalOperationFault"
7431 message="illegalOperationFault"/>
7432     </wsdl:operation>
7433
7434     <wsdl:operation name="batchSuspendUntil">
7435         <wsdl:input message="batchSuspendUntil"/>
7436         <wsdl:output message="batchSuspendUntilResponse"/>
7437     </wsdl:operation>
7438
7439     <wsdl:operation name="updateComment">
7440         <wsdl:input message="updateComment"/>
7441         <wsdl:output message="updateCommentResponse"/>
7442         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7443         <wsdl:fault name="illegalArgumentFault"
7444 message="illegalArgumentFault"/>
7445         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7446         <wsdl:fault name="illegalOperationFault"
7447 message="illegalOperationFault"/>
7448     </wsdl:operation>
7449
7450     <wsdl:operation name="getMyTaskAbstracts">
7451         <wsdl:input message="getMyTaskAbstracts"/>
7452         <wsdl:output message="getMyTaskAbstractsResponse"/>
7453         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7454         <wsdl:fault name="illegalArgumentFault"
7455 message="illegalArgumentFault"/>
7456         <wsdl:fault name="illegalOperationFault"
7457 message="illegalOperationFault"/>
7458     </wsdl:operation>
7459
7460     <wsdl:operation name="getMyTaskDetails">
7461         <wsdl:input message="getMyTaskDetails"/>
7462         <wsdl:output message="getMyTaskDetailsResponse"/>
7463         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7464         <wsdl:fault name="illegalArgumentFault"
7465 message="illegalArgumentFault"/>
7466         <wsdl:fault name="illegalOperationFault"
7467 message="illegalOperationFault"/>
7468     </wsdl:operation>
7469
7470     <wsdl:operation name="query">
7471         <wsdl:input message="query"/>
7472         <wsdl:output message="queryResponse"/>
7473         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7474         <wsdl:fault name="illegalArgumentFault"
7475 message="illegalArgumentFault"/>
7476     </wsdl:operation>
7477
7478     <wsdl:operation name="activate">
7479         <wsdl:input message="activate"/>
7480         <wsdl:output message="activateResponse"/>
7481         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7482         <wsdl:fault name="illegalArgumentFault"
7483 message="illegalArgumentFault"/>
7484         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7485         <wsdl:fault name="illegalOperationFault"
7486 message="illegalOperationFault"/>

```



```

7487     </wsdl:operation>
7488
7489     <wsdl:operation name="batchActivate">
7490         <wsdl:input message="batchActivate"/>
7491         <wsdl:output message="batchActivateResponse"/>
7492     </wsdl:operation>
7493
7494     <wsdl:operation name="nominate">
7495         <wsdl:input message="nominate"/>
7496         <wsdl:output message="nominateResponse"/>
7497         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7498         <wsdl:fault name="illegalArgumentFault"
7499 message="illegalArgumentFault"/>
7500         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7501         <wsdl:fault name="illegalOperationFault"
7502 message="illegalOperationFault"/>
7503     </wsdl:operation>
7504
7505     <wsdl:operation name="batchNominate">
7506         <wsdl:input message="batchNominate"/>
7507         <wsdl:output message="batchNominateResponse"/>
7508     </wsdl:operation>
7509
7510     <wsdl:operation name="setGenericHumanRole">
7511         <wsdl:input message="setGenericHumanRole"/>
7512         <wsdl:output message="setGenericHumanRoleResponse"/>
7513         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7514         <wsdl:fault name="illegalArgumentFault"
7515 message="illegalArgumentFault"/>
7516         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7517         <wsdl:fault name="illegalOperationFault"
7518 message="illegalOperationFault"/>
7519     </wsdl:operation>
7520
7521     <wsdl:operation name="batchSetGenericHumanRole">
7522         <wsdl:input message="batchSetGenericHumanRole"/>
7523         <wsdl:output message="batchSetGenericHumanRoleResponse"/>
7524     </wsdl:operation>
7525
7526 </wsdl:portType>
7527 </wsdl:definitions>

```

7528

E. WS-HumanTask Parent API Port Type

```

7529 <?xml version="1.0" encoding="UTF-8"?>
7530 <!--
7531 Copyright (c) OASIS Open 2009. All Rights Reserved.
7532 -->
7533 <wsdl:definitions
7534     targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
7535     humantask/leantask/api/200803"
7536     xmlns="http://docs.oasis-open.org/ns/bpel4people/ws-
7537     humantask/leantask/api/200803"
7538     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
7539     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
7540     xmlns:htd="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803"
7541     xmlns:htt="http://docs.oasis-open.org/ns/bpel4people/ws-
7542     humantask/types/200803">
7543
7544     <wsdl:documentation>
7545         Web Service Definition for WS-HumanTask 1.1 - Operations for Task Parent
7546 Applications
7547     </wsdl:documentation>
7548
7549     <wsdl:types>
7550     <xsd:schema
7551         targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
7552     humantask/leantask/api/200803"
7553         elementFormDefault="qualified"
7554         blockDefault="#all">
7555
7556         <xsd:import
7557             namespace="http://docs.oasis-open.org/ns/bpel4people/ws-
7558     humantask/200803"
7559             schemaLocation="ws-humantask.xsd"/>
7560         <xsd:import
7561             namespace="http://docs.oasis-open.org/ns/bpel4people/ws-
7562     humantask/types/200803"
7563             schemaLocation="ws-humantask-types.xsd"/>
7564
7565         <!-- Input and output elements -->
7566         <xsd:element name="registerLeanTaskDefinition">
7567             <xsd:complexType>
7568                 <xsd:sequence>
7569                     <xsd:element name="taskDefinition" type="htd:tLeanTask" />
7570                 </xsd:sequence>
7571             </xsd:complexType>
7572         </xsd:element>
7573         <xsd:element name="registerLeanTaskDefinitionResponse">
7574             <xsd:complexType>
7575                 <xsd:sequence>
7576                     <xsd:element name="taskName" type="xsd:NCName" />
7577                 </xsd:sequence>
7578             </xsd:complexType>
7579         </xsd:element>
7580
7581         <xsd:element name="unregisterLeanTaskDefinition">

```



```

7582     <xsd:complexType>
7583         <xsd:sequence>
7584             <xsd:element name="taskName" type="xsd:NCName" />
7585         </xsd:sequence>
7586     </xsd:complexType>
7587 </xsd:element>
7588 <xsd:element name="unregisterLeanTaskDefinitionResponse">
7589     <xsd:complexType>
7590         <xsd:sequence>
7591             <xsd:element name="taskName" type="xsd:NCName" />
7592         </xsd:sequence>
7593     </xsd:complexType>
7594 </xsd:element>
7595
7596 <xsd:element name="listLeanTaskDefinitions">
7597     <xsd:complexType>
7598         <xsd:sequence>
7599             <xsd:annotation>
7600                 <xsd:documentation>Empty message</xsd:documentation>
7601             </xsd:annotation>
7602         </xsd:sequence>
7603     </xsd:complexType>
7604 </xsd:element>
7605 <xsd:element name="listLeanTaskDefinitionsResponse">
7606     <xsd:complexType>
7607         <xsd:sequence>
7608             <xsd:element name="leanTaskDefinitions">
7609                 <xsd:complexType>
7610                     <xsd:sequence>
7611                         <xsd:element name="leanTaskDefinition" type="htd:tLeanTask"
7612 minOccurs="0" maxOccurs="unbounded" />
7613                     </xsd:sequence>
7614                 </xsd:complexType>
7615             </xsd:element>
7616         </xsd:sequence>
7617     </xsd:complexType>
7618 </xsd:element>
7619
7620 <xsd:element name="createLeanTask">
7621     <xsd:complexType>
7622         <xsd:sequence>
7623             <xsd:element name="inputMessage">
7624                 <xsd:complexType>
7625                     <xsd:sequence>
7626                         <xsd:any processContents="lax" namespace="##any" />
7627                     </xsd:sequence>
7628                 </xsd:complexType>
7629             </xsd:element>
7630             <xsd:element name="taskDefinition" type="htd:tLeanTask"
7631 minOccurs="0"/>
7632             <xsd:element name="taskName" type="xsd:NCName" minOccurs="0" />
7633         </xsd:sequence>
7634     </xsd:complexType>
7635 </xsd:element>
7636 <xsd:element name="createLeanTaskResponse">
7637     <xsd:complexType>
7638         <xsd:sequence>
7639             <xsd:element name="outputMessage">

```

```

7640         <xsd:complexType>
7641             <xsd:sequence>
7642                 <xsd:any processContents="lax" namespace="##any" />
7643             </xsd:sequence>
7644         </xsd:complexType>
7645     </xsd:element>
7646 </xsd:sequence>
7647 </xsd:complexType>
7648 </xsd:element>
7649
7650 <xsd:element name="createLeanTaskAsync">
7651     <xsd:complexType>
7652         <xsd:sequence>
7653             <xsd:element name="inputMessage">
7654                 <xsd:complexType>
7655                     <xsd:sequence>
7656                         <xsd:any processContents="lax" namespace="##any" />
7657                     </xsd:sequence>
7658                 </xsd:complexType>
7659             </xsd:element>
7660             <xsd:element name="taskDefinition" type="htd:tLeanTask"
7661 minOccurs="0"/>
7662             <xsd:element name="taskName" type="xsd:NCName" minOccurs="0" />
7663         </xsd:sequence>
7664     </xsd:complexType>
7665 </xsd:element>
7666 <xsd:element name="createLeanTaskAsyncResponse">
7667     <xsd:complexType>
7668         <xsd:sequence>
7669             <xsd:annotation>
7670                 <xsd:documentation>Empty message</xsd:documentation>
7671             </xsd:annotation>
7672         </xsd:sequence>
7673     </xsd:complexType>
7674 </xsd:element>
7675
7676 <xsd:element name="createLeanTaskAsyncCallback">
7677     <xsd:complexType>
7678         <xsd:sequence>
7679             <xsd:element name="outputMessage">
7680                 <xsd:complexType>
7681                     <xsd:sequence>
7682                         <xsd:any processContents="lax" namespace="##any" />
7683                     </xsd:sequence>
7684                 </xsd:complexType>
7685             </xsd:element>
7686         </xsd:sequence>
7687     </xsd:complexType>
7688 </xsd:element>
7689
7690 <!-- Fault elements -->
7691 <xsd:element name="illegalState">
7692     <xsd:complexType>
7693         <xsd:sequence>
7694             <xsd:element name="status" type="htt:tStatus"/>
7695             <xsd:element name="message" type="xsd:string"/>
7696         </xsd:sequence>
7697     </xsd:complexType>

```

```

7698     </xsd:element>
7699
7700     <xsd:element name="illegalArgument" type="xsd:string"/>
7701
7702     <xsd:element name="illegalAccess" type="xsd:string"/>
7703
7704     </xsd:schema>
7705 </wsdl:types>
7706
7707 <!-- Declaration of messages -->
7708 <wsdl:message name="registerLeanTaskDefinition">
7709   <wsdl:part name="registerLeanTaskDefinition"
7710 element="registerLeanTaskDefinition"/>
7711 </wsdl:message>
7712 <wsdl:message name="registerLeanTaskDefinitionResponse">
7713   <wsdl:part name="registerLeanTaskDefinitionResponse"
7714 element="registerLeanTaskDefinitionResponse"/>
7715 </wsdl:message>
7716
7717 <wsdl:message name="unregisterLeanTaskDefinition">
7718   <wsdl:part name="unregisterLeanTaskDefinition"
7719 element="unregisterLeanTaskDefinition"/>
7720 </wsdl:message>
7721 <wsdl:message name="unregisterLeanTaskDefinitionResponse">
7722   <wsdl:part name="unregisterLeanTaskDefinitionResponse"
7723 element="unregisterLeanTaskDefinitionResponse"/>
7724 </wsdl:message>
7725
7726 <wsdl:message name="listLeanTaskDefinitions">
7727   <wsdl:part name="listLeanTaskDefinitions"
7728 element="listLeanTaskDefinitions"/>
7729 </wsdl:message>
7730 <wsdl:message name="listLeanTaskDefinitionsResponse">
7731   <wsdl:part name="listLeanTaskDefinitionsResponse"
7732 element="listLeanTaskDefinitionsResponse"/>
7733 </wsdl:message>
7734
7735 <wsdl:message name="createLeanTask">
7736   <wsdl:part name="createLeanTask" element="createLeanTask"/>
7737 </wsdl:message>
7738 <wsdl:message name="createLeanTaskResponse">
7739   <wsdl:part name="createLeanTaskResponse"
7740 element="createLeanTaskResponse"/>
7741 </wsdl:message>
7742
7743 <wsdl:message name="createLeanTaskAsync">
7744   <wsdl:part name="createLeanTaskAsync" element="createLeanTaskAsync"/>
7745 </wsdl:message>
7746 <wsdl:message name="createLeanTaskAsyncResponse">
7747   <wsdl:part name="createLeanTaskAsyncResponse"
7748 element="createLeanTaskAsyncResponse"/>
7749 </wsdl:message>
7750
7751 <wsdl:message name="createLeanTaskAsyncCallback">
7752   <wsdl:part name="createLeanTaskAsyncCallback"
7753 element="createLeanTaskAsyncCallback"/>
7754 </wsdl:message>
7755

```

```

7756 <!-- Declaration of fault messages -->
7757 <wsdl:message name="illegalStateFault">
7758   <wsdl:part name="illegalState" element="illegalState"/>
7759 </wsdl:message>
7760 <wsdl:message name="illegalArgumentFault">
7761   <wsdl:part name="illegalArgument" element="illegalArgument"/>
7762 </wsdl:message>
7763 <wsdl:message name="illegalAccessFault">
7764   <wsdl:part name="illegalAccess" element="illegalAccess"/>
7765 </wsdl:message>
7766
7767 <!-- Port type definitions -->
7768 <wsdl:portType name="leanTaskOperations">
7769
7770   <wsdl:operation name="registerLeanTaskDefinition">
7771     <wsdl:input message="registerLeanTaskDefinition"/>
7772     <wsdl:output message="registerLeanTaskDefinitionResponse"/>
7773     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7774     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7775   </wsdl:operation>
7776
7777   <wsdl:operation name="unregisterLeanTaskDefinition">
7778     <wsdl:input message="unregisterLeanTaskDefinition"/>
7779     <wsdl:output message="unregisterLeanTaskDefinitionResponse"/>
7780     <wsdl:fault name="illegalArgumentFault"
7781 message="illegalArgumentFault"/>
7782     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7783   </wsdl:operation>
7784
7785   <wsdl:operation name="listLeanTaskDefinitions">
7786     <wsdl:input message="listLeanTaskDefinitions"/>
7787     <wsdl:output message="listLeanTaskDefinitionsResponse"/>
7788     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7789   </wsdl:operation>
7790
7791   <wsdl:operation name="createLeanTask">
7792     <wsdl:input message="createLeanTask"/>
7793     <wsdl:output message="createLeanTaskResponse"/>
7794     <wsdl:fault name="illegalArgumentFault"
7795 message="illegalArgumentFault"/>
7796     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7797   </wsdl:operation>
7798
7799   <wsdl:operation name="createLeanTaskAsync">
7800     <wsdl:input message="createLeanTaskAsync"/>
7801     <wsdl:output message="createLeanTaskAsyncResponse"/>
7802     <wsdl:fault name="illegalArgumentFault"
7803 message="illegalArgumentFault"/>
7804     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7805   </wsdl:operation>
7806
7807 </wsdl:portType>
7808
7809 <wsdl:portType name="leanTaskCallbackOperations">
7810
7811   <wsdl:operation name="createLeanTaskAsyncCallback">
7812     <wsdl:input message="createLeanTaskAsyncCallback"/>
7813   </wsdl:operation>

```

```
7814
7815     </wsdl:portType>
7816
7817 </wsdl:definitions>
```

7818

F. WS-HumanTask Protocol Handler Port Types

```
7819 <?xml version="1.0" encoding="UTF-8"?>
7820 <!--
7821 Copyright (c) OASIS Open 2009. All Rights Reserved.
7822 -->
7823 <wsdl:definitions
7824   targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
7825 humantask/protocol/200803"
7826   xmlns="http://docs.oasis-open.org/ns/bpel4people/ws-
7827 humantask/protocol/200803"
7828   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
7829   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
7830   xmlns:htp="http://docs.oasis-open.org/ns/bpel4people/ws-
7831 humantask/protocol/200803">
7832
7833   <wsdl:documentation>
7834     Web Service Definition for WS-HumanTask 1.1 - Operations WS-HumanTask
7835 Protocol Participants
7836   </wsdl:documentation>
7837
7838   <wsdl:types>
7839     <xsd:schema
7840       targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
7841 humantask/protocol/200803"
7842       elementFormDefault="qualified"
7843       blockDefault="#all">
7844
7845       <xsd:complexType name="tProtocolMsgType">
7846         <xsd:sequence>
7847           <xsd:any namespace="##other" processContents="lax" minOccurs="0"
7848             maxOccurs="unbounded" />
7849         </xsd:sequence>
7850         <xsd:anyAttribute namespace="##any" processContents="lax" />
7851       </xsd:complexType>
7852
7853       <xsd:element name="skipped" type="htp:tProtocolMsgType" />
7854       <xsd:element name="fault" type="htp:tProtocolMsgType" />
7855       <xsd:element name="exit" type="htp:tProtocolMsgType" />
7856
7857       <xsd:element name="responseAction" type="xsd:anyURI" />
7858       <xsd:element name="responseOperation" type="xsd:NCName" />
7859     </xsd:schema>
7860   </wsdl:types>
7861
7862   <wsdl:message name="skipped">
7863     <wsdl:part name="parameters" element="skipped" />
7864   </wsdl:message>
7865   <wsdl:message name="fault">
7866     <wsdl:part name="parameters" element="fault" />
7867   </wsdl:message>
7868   <wsdl:message name="exit">
7869     <wsdl:part name="parameters" element="exit" />
7870   </wsdl:message>
7871
```

```
7872
7873 <wsdl:portType name="clientParticipantPortType">
7874   <wsdl:operation name="skippedOperation">
7875     <wsdl:input message="skipped" />
7876   </wsdl:operation>
7877   <wsdl:operation name="faultOperation">
7878     <wsdl:input message="fault" />
7879   </wsdl:operation>
7880 </wsdl:portType>
7881
7882 <wsdl:portType name="humanTaskParticipantPortType">
7883   <wsdl:operation name="exitOperation">
7884     <wsdl:input message="exit" />
7885   </wsdl:operation>
7886 </wsdl:portType>
7887
7888 </wsdl:definitions>
```

7889

G. WS-HumanTask Context Schema

```

7890 <?xml version="1.0" encoding="UTF-8"?>
7891 <!--
7892 Copyright (c) OASIS Open 2009. All Rights Reserved.
7893 -->
7894 <xsd:schema
7895   targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
7896   humantask/context/200803"
7897   xmlns="http://docs.oasis-open.org/ns/bpel4people/ws-
7898   humantask/context/200803"
7899   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
7900   xmlns:htt="http://docs.oasis-open.org/ns/bpel4people/ws-
7901   humantask/types/200803"
7902   elementFormDefault="qualified"
7903   blockDefault="#all">
7904
7905   <xsd:annotation>
7906     <xsd:documentation>
7907       XML Schema for WS-HumanTask 1.1 - Human Task Context for Task
7908       Interactions
7909     </xsd:documentation>
7910   </xsd:annotation>
7911
7912   <!-- other namespaces -->
7913   <xsd:import
7914     namespace="http://www.w3.org/XML/1998/namespace"
7915     schemaLocation="http://www.w3.org/2001/xml.xsd"/>
7916   <xsd:import
7917     namespace="http://docs.oasis-open.org/ns/bpel4people/ws-
7918     humantask/types/200803"
7919     schemaLocation="ws-humantask-types.xsd"/>
7920
7921   <!-- human task context -->
7922   <xsd:element name="humanTaskRequestContext"
7923   type="tHumanTaskRequestContext"/>
7924   <xsd:complexType name="tHumanTaskRequestContext">
7925     <xsd:complexContent>
7926       <xsd:extension base="tHumanTaskContextBase">
7927         <xsd:sequence>
7928           <xsd:element name="peopleAssignments" type="tPeopleAssignments"
7929   minOccurs="0"/>
7930           <xsd:element name="isSkipable" type="xsd:boolean" minOccurs="0"/>
7931           <xsd:element name="expirationTime" type="xsd:dateTime"
7932   minOccurs="0"/>
7933           <xsd:element name="activationDeferralTime" type="xsd:dateTime"
7934   minOccurs="0"/>
7935           <xsd:any namespace="##other" processContents="lax" minOccurs="0"
7936   maxOccurs="unbounded"/>
7937         </xsd:sequence>
7938       </xsd:extension>
7939     </xsd:complexContent>
7940   </xsd:complexType>
7941   <xsd:element name="humanTaskResponseContext"
7942   type="tHumanTaskResponseContext"/>

```



```

7943 <xsd:complexType name="tHumanTaskResponseContext">
7944 <xsd:complexContent>
7945 <xsd:extension base="tHumanTaskContextBase">
7946 <xsd:sequence>
7947 <xsd:element name="actualOwner" type="htt:tUser"/>
7948 <xsd:element name="actualPeopleAssignments"
7949 type="tPeopleAssignments"/>
7950 <xsd:element name="outcome" type="xsd:string" minOccurs="0"/>
7951 <xsd:any namespace="##other" processContents="lax" minOccurs="0"
7952 maxOccurs="unbounded"/>
7953 </xsd:sequence>
7954 </xsd:extension>
7955 </xsd:complexContent>
7956 </xsd:complexType>
7957 <xsd:complexType name="tHumanTaskContextBase" abstract="true">
7958 <xsd:sequence>
7959 <xsd:element name="priority" type="htt:tPriority" minOccurs="0"/>
7960 <xsd:element name="attachments" type="tAttachments" minOccurs="0"/>
7961 </xsd:sequence>
7962 </xsd:complexType>
7963
7964 <!-- people assignments -->
7965 <xsd:complexType name="tPeopleAssignments">
7966 <xsd:sequence>
7967 <xsd:element ref="genericHumanRole" minOccurs="0"
7968 maxOccurs="unbounded"/>
7969 </xsd:sequence>
7970 </xsd:complexType>
7971 <xsd:element name="genericHumanRole" type="tGenericHumanRole"
7972 abstract="true" block="restriction extension"/>
7973 <xsd:element name="potentialOwners" type="tGenericHumanRole"
7974 substitutionGroup="genericHumanRole"/>
7975 <xsd:element name="excludedOwners" type="tGenericHumanRole"
7976 substitutionGroup="genericHumanRole"/>
7977 <xsd:element name="taskInitiator" type="tGenericHumanRole"
7978 substitutionGroup="genericHumanRole"/>
7979 <xsd:element name="taskStakeholders" type="tGenericHumanRole"
7980 substitutionGroup="genericHumanRole"/>
7981 <xsd:element name="businessAdministrators" type="tGenericHumanRole"
7982 substitutionGroup="genericHumanRole"/>
7983 <xsd:element name="recipients" type="tGenericHumanRole"
7984 substitutionGroup="genericHumanRole"/>
7985 <xsd:complexType name="tGenericHumanRole">
7986 <xsd:sequence>
7987 <xsd:element ref="htt:organizationalEntity"/>
7988 </xsd:sequence>
7989 </xsd:complexType>
7990
7991 <!-- attachments -->
7992 <xsd:complexType name="tAttachments">
7993 <xsd:sequence>
7994 <xsd:element name="returnAttachments" type="tReturnAttachments"
7995 minOccurs="0"/>
7996 <xsd:element ref="htt:attachment" minOccurs="0" maxOccurs="unbounded"/>
7997 </xsd:sequence>
7998 </xsd:complexType>
7999 <xsd:simpleType name="tReturnAttachments">
8000 <xsd:restriction base="xsd:string">

```

```
8001     <xsd:enumeration value="all"/>
8002     <xsd:enumeration value="newOnly"/>
8003     <xsd:enumeration value="none"/>
8004     </xsd:restriction>
8005 </xsd:simpleType>
8006
8007 </xsd:schema>
```

8008

H. WS-HumanTask Policy Assertion Schema

```
8009 <?xml version="1.0" encoding="UTF-8"?>
8010 <!--
8011 Copyright (c) OASIS Open 2009. All Rights Reserved.
8012 -->
8013 <xsd:schema
8014   targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
8015 humantask/policy/200803"
8016   xmlns="http://docs.oasis-open.org/ns/bpel4people/ws-
8017 humantask/policy/200803"
8018   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
8019   xmlns:wsp="http://www.w3.org/ns/ws-policy"
8020   elementFormDefault="qualified"
8021   blockDefault="#all">
8022
8023   <xsd:annotation>
8024     <xsd:documentation>
8025       XML Schema for WS-HumanTask 1.1 - WS-HumanTask Policy Assertion
8026     </xsd:documentation>
8027   </xsd:annotation>
8028
8029   <!-- other namespaces -->
8030   <xsd:import
8031     namespace="http://www.w3.org/ns/ws-policy"
8032     schemaLocation="http://www.w3.org/2007/02/ws-policy.xsd" />
8033
8034   <!-- ws-humantask policy assertion -->
8035   <xsd:element name="HumanTaskAssertion" type="tHumanTaskAssertion"/>
8036   <xsd:complexType name="tHumanTaskAssertion" >
8037     <xsd:attribute ref="wsp:Optional" />
8038     <xsd:anyAttribute namespace="##any" processContents="lax" />
8039   </xsd:complexType>
8040
8041 </xsd:schema>
```

8042

I. Sample

8043

This appendix contains the full sample used in this specification.

8044

8045

WSDL Definition

8046

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

8047

```
<!--
```

8048

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

8049

```
-->
```

8050

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

8051

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

8052

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

8053

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

8054

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

8055

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

8056

```
<wsdl:documentation>
```

8058

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

8059

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

8060

```
<wsdl:types>
```

8061

```
<xsd:schema
```

8063

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

8064

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

8065

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

8066

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

8067

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

8068

```
<xsd:complexType>
```

8069

```
<xsd:sequence>
```

8070

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

8071

```
<xsd:complexType>
```

8072

```
<xsd:sequence>
```

8073

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

8074

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

8075

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

8076

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

8077

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

8078

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

8079

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

8080

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

8081

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

8082

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

8083

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

8084

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

8085

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

8086

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

8087

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

8088

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

8089

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

8090

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

8091

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

8093

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

8094

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

```

8095 </wsdl:message>
8096 <wsdl:message name="ClaimApprovalResponse">
8097   <wsdl:part name="ClaimApprovalResponse" type="xsd:boolean" />
8098 </wsdl:message>
8099 <wsdl:message name="notifyRequest">
8100   <wsdl:part name="firstname" type="xsd:string" />
8101   <wsdl:part name="lastname" type="xsd:string" />
8102 </wsdl:message>
8103
8104 <wsdl:portType name="ClaimsHandlingPT">
8105   <wsdl:operation name="approve">
8106     <wsdl:input message="tns:ClaimApprovalRequest" />
8107   </wsdl:operation>
8108   <wsdl:operation name="escalate">
8109     <wsdl:input message="tns:ClaimApprovalRequest" />
8110   </wsdl:operation>
8111 </wsdl:portType>
8112
8113 <wsdl:portType name="ClaimsHandlingCallbackPT">
8114   <wsdl:operation name="approvalResponse">
8115     <wsdl:input message="tns:ClaimApprovalResponse" />
8116   </wsdl:operation>
8117 </wsdl:portType>
8118
8119 <wsdl:portType name="ClaimApprovalReminderPT">
8120   <wsdl:operation name="notify">
8121     <wsdl:input message="tns:notifyRequest" />
8122   </wsdl:operation>
8123 </wsdl:portType>
8124
8125 </wsdl:definitions>

```

8126

8127 **Human Interaction Definition**

```

8128 <?xml version="1.0" encoding="UTF-8"?>
8129 <!--
8130 Copyright (c) OASIS Open 2009. All Rights Reserved.
8131 -->
8132 <htd:humanInteractions
8133   xmlns:htd="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803"
8134   xmlns:htt="http://docs.oasis-open.org/ns/bpel4people/ws-
8135   humantask/types/200803"
8136   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
8137   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
8138   xmlns:cl="http://www.example.com/claims/"
8139   xmlns:tns="http://www.example.com"
8140   targetNamespace="http://www.example.com"
8141   xsi:schemaLocation="http://docs.oasis-open.org/ns/bpel4people/ws-
8142   humantask/200803 ../../xml/ws-humantask.xsd">
8143
8144   <htd:documentation>
8145     Example for WS-HumanTask 1.1 - WS-HumanTask Task Definition
8146   </htd:documentation>
8147
8148   <htd:import importType="http://schemas.xmlsoap.org/wsdl/"
8149     location="ws-humantask-example-claim-approval.wsdl"
8150     namespace="http://www.example.com/claims/" />
8151

```

```

8152 <htd:logicalPeopleGroups>
8153
8154   <htd:logicalPeopleGroup name="regionalClerks">
8155     <htd:documentation xml:lang="en-US">
8156       The group of clerks responsible for the region specified.
8157     </htd:documentation>
8158     <htd:parameter name="region" type="xsd:string" />
8159   </htd:logicalPeopleGroup>
8160
8161   <htd:logicalPeopleGroup name="regionalManager">
8162     <htd:documentation xml:lang="en-US">
8163       The manager responsible for the region specified.
8164     </htd:documentation>
8165     <htd:parameter name="region" type="xsd:string" />
8166   </htd:logicalPeopleGroup>
8167
8168   <htd:logicalPeopleGroup name="clerksManager">
8169     <htd:documentation xml:lang="en-US">
8170       The manager of the clerk whose user ID is passed as parameter.
8171     </htd:documentation>
8172     <htd:parameter name="clerkUserID" type="xsd:string" />
8173   </htd:logicalPeopleGroup>
8174
8175   <htd:logicalPeopleGroup name="directorClaims">
8176     <htd:documentation xml:lang="en-US">
8177       The functional director responsible for claims processing.
8178     </htd:documentation>
8179   </htd:logicalPeopleGroup>
8180
8181 </htd:logicalPeopleGroups>
8182
8183 <htd:tasks>
8184
8185   <htd:task name="ApproveClaim">
8186     <htd:documentation xml:lang="en-US">
8187       This task is used to handle claims that require manual
8188       approval.
8189     </htd:documentation>
8190
8191     <htd:interface portType="cl:ClaimsHandlingPT"
8192       operation="approve"
8193       responsePortType="cl:ClaimsHandlingCallbackPT"
8194       responseOperation="approvalResponse" />
8195
8196     <htd:priority>
8197       htd:getInput("ClaimApprovalRequest")/prio
8198     </htd:priority>
8199
8200     <htd:peopleAssignments>
8201       <htd:potentialOwners>
8202         <htd:from logicalPeopleGroup="regionalClerks">
8203           <htd:argument name="region">
8204             htd:getInput("ClaimApprovalRequest")/region
8205           </htd:argument>
8206         </htd:from>
8207       </htd:potentialOwners>
8208
8209       <htd:businessAdministrators>

```

```

8210     <htd:from logicalPeopleGroup="regionalManager">
8211         <htd:argument name="region">
8212             htd:getInput("ClaimApprovalRequest")/region
8213         </htd:argument>
8214     </htd:from>
8215 </htd:businessAdministrators>
8216 </htd:peopleAssignments>
8217
8218 <htd:delegation potentialDelegates="nobody" />
8219
8220 <htd:presentationElements>
8221
8222     <htd:name xml:lang="en-US">Approve Claim</htd:name>
8223     <htd:name xml:lang="de-DE">
8224         Genehmigung der Schadensforderung
8225     </htd:name>
8226
8227     <htd:presentationParameters>
8228         <htd:presentationParameter name="firstname"
8229             type="xsd:string">
8230             htd:getInput("ClaimApprovalRequest")/cust/firstname
8231         </htd:presentationParameter>
8232         <htd:presentationParameter name="lastname"
8233             type="xsd:string">
8234             htd:getInput("ClaimApprovalRequest")/cust/lastname
8235         </htd:presentationParameter>
8236         <htd:presentationParameter name="euroAmount"
8237             type="xsd:double">
8238             htd:getInput("ClaimApprovalRequest")/amount
8239         </htd:presentationParameter>
8240     </htd:presentationParameters>
8241
8242     <htd:subject xml:lang="en-US">
8243         Approve the insurance claim for €$euroAmount$ on behalf of
8244         $firstname$ $lastname$
8245     </htd:subject>
8246     <htd:subject xml:lang="de-DE">
8247         Genehmigung der Schadensforderung über €$euroAmount$ für
8248         $firstname$ $lastname$
8249     </htd:subject>
8250
8251     <htd:description xml:lang="en-US" contentType="text/plain">
8252         Approve this claim following corporate guideline
8253         #4711.0815/7 ...
8254     </htd:description>
8255     <htd:description xml:lang="en-US" contentType="text/html">
8256         <![CDATA[
8257             <p>
8258                 Approve this claim following corporate guideline
8259                 <b>#4711.0815/7</b>
8260                 ...
8261             </p>
8262             ]]>
8263     </htd:description>
8264     <htd:description xml:lang="de-DE" contentType="text/plain">
8265         Genehmigen Sie diese Schadensforderung entsprechend
8266         Richtlinie Nr. 4711.0815/7 ...
8267     </htd:description>

```

```

8268 <htd:description xml:lang="de-DE" contentType="text/html">
8269 <![CDATA[
8270 <p>
8271     Genehmigen Sie diese Schadensforderung entsprechend
8272     Richtlinie
8273     <b>Nr. 4711.0815/7</b>
8274     ...
8275 </p>
8276 ]]>
8277 </htd:description>
8278
8279 </htd:presentationElements>
8280
8281
8282 <htd:deadlines>
8283
8284 <htd:startDeadline name="sendReminder">
8285 <htd:documentation xml:lang="en-US">
8286     If not started within 3 days, - escalation notifications
8287     are sent if the claimed amount is less than 10000 - to the
8288     task's potential owners to remind them or their todo - to
8289     the regional manager, if this approval is of high priority
8290     (0,1, or 2) - the task is reassigned to Alan if the
8291     claimed amount is greater than or equal 10000
8292 </htd:documentation>
8293 <htd:for>P3D</htd:for>
8294
8295 <htd:escalation name="reminder">
8296
8297 <htd:condition>
8298 <![CDATA[
8299     htd:getInput("ClaimApprovalRequest")/amount < 10000
8300 ]]>
8301 </htd:condition>
8302
8303 <htd:toParts>
8304 <htd:toPart name="firstname">
8305     htd:getInput("ClaimApprovalRequest","ApproveClaim")
8306     /firstname
8307 </htd:toPart>
8308 <htd:toPart name="lastname">
8309     htd:getInput("ClaimApprovalRequest","ApproveClaim")
8310     /lastname
8311 </htd:toPart>
8312 </htd:toParts>
8313
8314 <htd:localNotification
8315     reference="tns:ClaimApprovalReminder">
8316
8317 <htd:documentation xml:lang="en-US">
8318     Reuse the predefined notification
8319     "ClaimApprovalReminder". Overwrite the recipients with
8320     the task's potential owners.
8321 </htd:documentation>
8322
8323 <htd:peopleAssignments>
8324 <htd:recipients>
8325 <htd:from>

```



```

8326         htd:getPotentialOwners("ApproveClaim")
8327         </htd:from>
8328         </htd:recipients>
8329         </htd:peopleAssignments>
8330
8331     </htd:localNotification>
8332
8333 </htd:escalation>
8334
8335 <htd:escalation name="highPrio">
8336
8337     <htd:condition>
8338         <![CDATA[
8339             (htd:getInput("ClaimApprovalRequest")/amount < 10000
8340             && htd:getInput("ClaimApprovalRequest")/prio <= 2)
8341         ]]>
8342     </htd:condition>
8343
8344     <!-- task input implicitly passed to the notification -->
8345
8346     <htd:notification name="ClaimApprovalOverdue">
8347         <htd:documentation xml:lang="en-US">
8348             An inline defined notification using the approval data
8349             as its input.
8350         </htd:documentation>
8351
8352         <htd:interface portType="cl:ClaimsHandlingPT"
8353             operation="escalate" />
8354
8355         <htd:peopleAssignments>
8356             <htd:recipients>
8357                 <htd:from logicalPeopleGroup="regionalManager">
8358                     <htd:argument name="region">
8359                         htd:getInput("ClaimApprovalRequest")/region
8360                     </htd:argument>
8361                 </htd:from>
8362             </htd:recipients>
8363         </htd:peopleAssignments>
8364
8365         <htd:presentationElements>
8366             <htd:name xml:lang="en-US">
8367                 Claim approval overdue
8368             </htd:name>
8369             <htd:name xml:lang="de-DE">
8370                 Überfällige Schadensforderungsgenehmigung
8371             </htd:name>
8372         </htd:presentationElements>
8373
8374     </htd:notification>
8375
8376 </htd:escalation>
8377
8378 <htd:escalation name="highAmountReassign">
8379
8380     <htd:condition>
8381         <![CDATA[
8382             htd:getInput("ClaimApprovalRequest")/amount >= 10000
8383         ]]>

```

```

8384     </htd:condition>
8385
8386     <htd:reassignment>
8387         <htd:documentation>
8388             Reassign task to Alan if amount is greater than or
8389             equal 10000.
8390         </htd:documentation>
8391
8392         <htd:potentialOwners>
8393             <htd:from>
8394                 <htd:literal>
8395                     <htt:organizationalEntity>
8396                         <htt:user>Alan</htt:user>
8397                     </htt:organizationalEntity>
8398                 </htd:literal>
8399             </htd:from>
8400         </htd:potentialOwners>
8401
8402     </htd:reassignment>
8403
8404 </htd:escalation>
8405
8406 </htd:startDeadline>
8407
8408
8409 <htd:completionDeadline name="notifyManager">
8410     <htd:documentation xml:lang="en-US">
8411         When not completed within 3 hours after having been
8412         claimed, the manager of the clerk who claimed the activity
8413         is notified.
8414     </htd:documentation>
8415     <htd:for>PT3H</htd:for>
8416
8417     <htd:escalation name="delayedApproval">
8418
8419         <htd:notification name="ClaimApprovalOverdue">
8420             <htd:documentation xml:lang="en-US">
8421                 An inline defined notification using the approval data
8422                 as its input.
8423             </htd:documentation>
8424
8425             <htd:interface portType="cl:ClaimsHandlingPT"
8426                 operation="escalate" />
8427
8428             <htd:peopleAssignments>
8429                 <htd:recipients>
8430                     <htd:from logicalPeopleGroup="clerksManager">
8431                         <htd:argument name="clerkUserID">
8432                             htd:getActualOwner("ApproveClaim")
8433                         </htd:argument>
8434                     </htd:from>
8435                 </htd:recipients>
8436             </htd:peopleAssignments>
8437
8438             <htd:presentationElements>
8439                 <htd:name xml:lang="en-US">
8440                     Claim approval overdue
8441                 </htd:name>

```

```

8442         <htd:name xml:lang="de-DE">
8443             Überfällige Schadensforderungsgenehmigung
8444         </htd:name>
8445     </htd:presentationElements>
8446
8447 </htd:notification>
8448
8449 </htd:escalation>
8450 </htd:completionDeadline>
8451
8452 <htd:completionDeadline name="notifyDirector">
8453     <htd:documentation xml:lang="en-US">
8454         When not completed within 2 days after having been
8455         claimed, the functional director of claims processing is
8456         notified.
8457     </htd:documentation>
8458     <htd:for>P2D</htd:for>
8459
8460 <htd:escalation name="severelyDelayedApproval">
8461
8462     <htd:notification name="ClaimApprovalOverdue">
8463         <htd:documentation xml:lang="en-US">
8464             An inline defined notification using the approval data
8465             as its input.
8466         </htd:documentation>
8467
8468         <htd:interface portType="cl:ClaimsHandlingPT"
8469             operation="escalate" />
8470
8471         <htd:peopleAssignments>
8472             <htd:recipients>
8473                 <htd:from logicalPeopleGroup="directorClaims">
8474                     <htd:argument name="clerkUserID">
8475                         htd:getActualOwner("ApproveClaim")
8476                     </htd:argument>
8477                 </htd:from>
8478             </htd:recipients>
8479         </htd:peopleAssignments>
8480
8481         <htd:presentationElements>
8482             <htd:name xml:lang="en-US">
8483                 Claim approval severely overdue
8484             </htd:name>
8485             <htd:name xml:lang="de-DE">
8486                 Hochgradig überfällige Schadensforderungsgenehmigung
8487             </htd:name>
8488         </htd:presentationElements>
8489     </htd:notification>
8490
8491 </htd:escalation>
8492 </htd:completionDeadline>
8493
8494 </htd:deadlines>
8495
8496 </htd:task>
8497
8498 </htd:tasks>

```

```

8500
8501 <htd:notifications>
8502
8503 <htd:notification name="ClaimApprovalReminder">
8504 <htd:documentation xml:lang="en-US">
8505     This notification is used to remind people of pending
8506     out-dated claim approvals. Recipients of this notification
8507     maybe overridden when it is referenced.
8508 </htd:documentation>
8509
8510 <htd:interface portType="cl:ClaimApprovalReminderPT"
8511     operation="notify" />
8512
8513 <htd:peopleAssignments>
8514 <htd:recipients>
8515 <htd:from>
8516 <htd:literal>
8517 <htt:organizationalEntity>
8518 <htt:user>Alan</htt:user>
8519 <htt:user>Dieter</htt:user>
8520 <htt:user>Frank</htt:user>
8521 <htt:user>Gerhard</htt:user>
8522 <htt:user>Ivana</htt:user>
8523 <htt:user>Karsten</htt:user>
8524 <htt:user>Matthias</htt:user>
8525 <htt:user>Patrick</htt:user>
8526 </htt:organizationalEntity>
8527 </htd:literal>
8528 </htd:from>
8529 </htd:recipients>
8530 </htd:peopleAssignments>
8531
8532 <htd:presentationElements>
8533
8534 <htd:name xml:lang="en-US">Approve Claim</htd:name>
8535 <htd:name xml:lang="de-DE">
8536     Genehmigung der Schadensforderung
8537 </htd:name>
8538
8539 <htd:presentationParameters>
8540 <htd:presentationParameter name="firstname"
8541     type="xsd:string">
8542     htd:getInput("firstname")
8543 </htd:presentationParameter>
8544 <htd:presentationParameter name="lastname"
8545     type="xsd:string">
8546     htd:getInput("lastname")
8547 </htd:presentationParameter>
8548 <htd:presentationParameter name="id" type="xsd:string">
8549     htd:getInput("taskId")
8550 </htd:presentationParameter>
8551 </htd:presentationParameters>
8552
8553 <htd:subject xml:lang="en-US">
8554     Claim approval for $firstname$, $lastname$ is overdue. See
8555     task $id$.
8556 </htd:subject>
8557

```

```
8558     </htd:presentationElements>
8559
8560     </htd:notification>
8561
8562     </htd:notifications>
8563
8564 </htd:humanInteractions>
```

8565 J. Acknowledgements

8566 The following individuals have participated in the creation of this specification and are gratefully
8567 acknowledged:

8568

8569 **Members of the BPEL4People Technical Committee:**

8570 Phillip Allen, Microsoft Corporation

8571 Ashish Agrawal, Adobe Systems

8572 Mike Amend, BEA Systems, Inc.

8573 Stefan Baeuerle, SAP AG

8574 Charlton Barreto, Adobe Systems

8575 Justin Brunt, TIBCO Software Inc.

8576 Martin Chapman, Oracle Corporation

8577 Luc Clément, Active Endpoints, Inc.

8578 Manoj Das, Oracle Corporation

8579 Alireza Farhoush, TIBCO Software Inc.

8580 Mark Ford, Active Endpoints, Inc.

8581 Sabine Holz, SAP AG

8582 Dave Ings, IBM

8583 Gershon Janssen, Individual

8584 Diane Jordan, IBM

8585 Anish Karmarkar, Oracle Corporation

8586 Ulrich Keil, SAP AG

8587 Oliver Kieselbach, SAP AG

8588 Matthias Kloppmann, IBM

8589 Dieter König, IBM

8590 Marita Kruempelmann, SAP AG

8591 Frank Leymann, IBM

8592 Mark Little, Red Hat

8593 Alexander Malek, Microsoft Corporation

8594 Ashok Malhotra, Oracle Corporation

8595 Mike Marin, IBM

8596 Vinkesh Mehta, Deloitte Consulting LLP

8597 Jeff Mischkinisky, Oracle Corporation

8598 Ralf Mueller, Oracle Corporation

8599 Krasimir Nedkov, SAP AG

8600 Benjamin Notheis, SAP AG

8601 Michael Pellegrini, Active Endpoints, Inc.

8602 Hannah Petereit, SAP AG

8603 Gerhard Pfau, IBM

8604 Karsten Ploesser, SAP AG

8605 Ravi Rangaswamy, Oracle Corporation
8606 Alan Rickayzen, SAP AG
8607 Michael Rowley, BEA Systems, Inc.
8608 Ron Ten-Hove, Sun Microsystems
8609 Ivana Trickovic, SAP AG
8610 Alessandro Triglia, OSS Nokalva
8611 Claus von Riegen, SAP AG
8612 Peter Walker, Sun Microsystems
8613 Franz Weber, SAP AG
8614 Prasad Yendluri, Software AG, Inc.

8615

8616 **WS-HumanTask 1.0 Specification Contributors:**

8617 Ashish Agrawal, Adobe
8618 Mike Amend, BEA
8619 Manoj Das, Oracle
8620 Mark Ford, Active Endpoints
8621 Chris Keller, Active Endpoints
8622 Matthias Kloppmann, IBM
8623 Dieter König, IBM
8624 Frank Leymann, IBM
8625 Ralf Müller, Oracle
8626 Gerhard Pfau, IBM
8627 Karsten Plösser, SAP
8628 Ravi Rangaswamy, Oracle
8629 Alan Rickayzen, SAP
8630 Michael Rowley, BEA
8631 Patrick Schmidt, SAP
8632 Ivana Trickovic, SAP
8633 Alex Yiu, Oracle
8634 Matthias Zeller, Adobe

8635

8636 The following individuals have provided valuable input into the design of this specification: Dave Ings,
8637 Diane Jordan, Mohan Kamath, Ulrich Keil, Matthias Kruse, Kurt Lind, Jeff Mischkinsky, Bhagat Nainani,
8638 Michael Pellegrini, Lars Rueter, Frank Ryan, David Shaffer, Will Stallard, Cyrille Waguët, Franz Weber,
8639 and Eric Wittmann.

K. Revision History

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:)

Revision	Date	Editor	Changes Made
			Issue 55, section 2.5.2 (import type xsd) 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

Revision	Date	Editor	Changes Made
CD-05-rev1	2009-07-15	Luc Clément	Issue 117
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
CD-08	2010-04-14	Luc Clément	CD08
CD-09	2010-04-14	Luc Clément	CD09 / PRD-03
PRD-03	2010-05-12	Luc Clément	PRD-03 Approved for Public Review
CD-09-rev1	2010-06-10	Dieter König	Issue 145
CD-10	2010-06-23	Luc Clément	CD10 / PRD-04