



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

## Committee Draft 09 / Public Review Draft 03

14 April 2010

### Specification URIs:

#### This 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> (Authoritative format)  
<http://docs.oasis-open.org/bpel4people/ws-humantask-1.1-spec-cd-09.pdf>

#### Previous Version:

<http://docs.oasis-open.org/bpel4people/ws-humantask-1.1-spec-cd-08.html>  
<http://docs.oasis-open.org/bpel4people/ws-humantask-1.1-spec-cd-08.doc>  
<http://docs.oasis-open.org/bpel4people/ws-humantask-1.1-spec-cd-08.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.....                                  | 9  |
| 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 .....  | 45  |
| 4.8.1  | Completion Conditions.....   | 46  |
| 4.8.2  | Result Construction from Parallel Subtasks.....                        | 47  |
| 4.9    | Elements for Handling Timeouts and Escalations .....                   | 51  |
| 4.10   | Human Task Behavior and State Transitions .....                        | 58  |
| 4.10.1 | Normal processing of a Human Task.....                                 | 58  |
| 4.10.2 | Releasing a Human Task .....   | 59  |
| 4.10.3 | Delegating or Forwarding a Human Task .....                            | 59  |
| 4.10.4 | Sub Task Event Propagation .....                                       | 59  |
| 4.11   | History of a Human Task.....   | 60  |
| 4.11.1 | Task Event Types and Data .....  | 61  |
| 4.11.2 | Retrieving the History .....   | 63  |
| 5      | Lean Tasks .....   | 66  |
| 5.1    | Overall Syntax .....   | 66  |
| 5.2    | Properties.....  | 66  |
| 5.3    | Message Schema.....  | 66  |
| 5.4    | Example: ToDoTask.....   | 68  |
| 6      | Notifications .....  | 69  |
| 6.1    | Overall Syntax .....   | 69  |
| 6.2    | Properties.....  | 70  |
| 6.3    | Notification Behavior and State Transitions.....                       | 70  |
| 7      | Programming Interfaces .....   | 71  |
| 7.1    | Operations for Client Applications .....                               | 71  |
| 7.1.1  | Participant Operations .....   | 71  |
| 7.1.2  | Simple Query Operations .....  | 83  |
| 7.1.3  | Advanced Query Operation .....   | 86  |
| 7.1.4  | Administrative Operations.....   | 90  |
| 7.1.5  | Operation Authorizations .....   | 91  |
| 7.2    | XPath Extension Functions .....  | 93  |
| 8      | Interoperable Protocol for Advanced Interaction with Human Tasks ..... | 100 |
| 8.1    | Human Task Coordination Protocol Messages .....                        | 102 |
| 8.2    | Protocol Messages.....   | 103 |
| 8.2.1  | Protocol Messages Received by a Task Parent .....                      | 103 |
| 8.2.2  | Protocol Messages Received by a Task.....                              | 103 |
| 8.3    | WSDL of the Protocol Endpoints.....                                    | 103 |
| 8.3.1  | Protocol Endpoint of the Task Parent .....                             | 103 |
| 8.3.2  | Protocol Endpoint of the Task.....                                     | 104 |
| 8.4    | Providing Human Task Context.....                                      | 104 |
| 8.4.1  | SOAP Binding of Human Task Context .....                               | 104 |

|       |   |     |
|-------|---|-----|
| 8.4.2 | Overriding Task Definition People Assignments .....   | 105 |
| 8.5   | Human Task Policy Assertion .....                     | 106 |
| 9     | Task Parent Interactions with Lean Tasks .....        | 107 |
| 9.1   | Operations for Task Parent Applications .....         | 107 |
| 9.2   | Lean Task Interactions .....                          | 107 |
| 9.2.1 | Register a Lean Task Definition .....                 | 107 |
| 9.2.2 | Unregister a Lean Task Definition .....               | 108 |
| 9.2.3 | List Lean Task Definitions .....                      | 108 |
| 9.2.4 | Create a Lean Task .....                              | 109 |
| 9.2.5 | Endpoints for Lean Task Operations .....              | 110 |
| 10    | Providing Callback Information for Human Tasks .....  | 112 |
| 10.1  | EPR Information Model Extension .....                 | 112 |
| 10.2  | XML Infoset Representation .....                      | 112 |
| 10.3  | Message Addressing Properties .....                   | 114 |
| 10.4  | SOAP Binding .....                                    | 115 |
| 11    | Security Considerations .....                         | 118 |
| 12    | Conformance .....                                     | 119 |
| A.    | Portability and Interoperability Considerations ..... | 120 |
| B.    | WS-HumanTask Language Schema .....                    | 121 |
| C.    | WS-HumanTask Data Types Schema .....                  | 136 |
| D.    | WS-HumanTask Client API Port Type .....               | 145 |
| E.    | WS-HumanTask Parent API Port Type .....               | 189 |
| F.    | WS-HumanTask Protocol Handler Port Types .....        | 195 |
| G.    | WS-HumanTask Context Schema .....                     | 197 |
| H.    | WS-HumanTask Policy Assertion Schema .....            | 200 |
| I.    | Sample .....  | 201 |
| J.    | Acknowledgements .....                                | 211 |
| K.    | Revision History .....                                | 213 |

---

# 1 Introduction

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

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

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

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

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

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

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

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

## 1.1 Terminology

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

## 46 1.2 Normative References

### 47 [RFC 1766]

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

### 50 [RFC 2046]

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

### 54 [RFC 2119]

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

### 57 [RFC 2396]

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

### 60 [RFC 3066]

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

### 63 [WSDL 1.1]

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

### 66 [WS-Addr-Core]

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

### 69 [WS-Addr-SOAP]

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

### 72 [WS-Addr-WSDL]

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

### 75 [WS-C]

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

### 78 [WS-Policy]

79 Web Services Policy 1.5 - Framework, W3C ~~Candidate~~ Recommendation ~~30 March~~ 04 September  
80 2007, available via <http://www.w3.org/TR/ws-policy/>

### 81 [WS-PolAtt]

82 Web Services Policy 1.5 - Attachment, W3C ~~Candidate~~ Recommendation ~~30 March~~ 04 September  
83 2007, available via [http://www.w3.org/TR/2007/CR-ws-policy-attach-](http://www.w3.org/TR/2007/CR-ws-policy-attach-20070330/)  
84 [20070330/](http://www.w3.org/TR/2007/CR-ws-policy-attach-20070330/) <http://www.w3.org/TR/ws-policy-attach/>

### 85 [XML Infoset]

86 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/)  
87 [infoset-20011024/](http://www.w3.org/TR/2001/REC-xml-infoset-20011024/)

### 88 [XML Namespaces]

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

91

- 92 |
- 93 **[XML Schema Part 1]**
- 94 XML Schema Part 1: Structures, W3C Recommendation, October 2004, available via
- 95 <http://www.w3.org/TR/xmlschema-1/>
- 96 **[XML Schema Part 2]**
- 97 XML Schema Part 2: Datatypes, W3C Recommendation, October 2004, available via
- 98 <http://www.w3.org/TR/xmlschema-2/>
- 99 **[XMLSpec]**
- 100 XML Specification, W3C Recommendation, February 1998, available via
- 101 <http://www.w3.org/TR/1998/REC-xml-19980210>
- 102 **[XPath 1.0]**
- 103 XML Path Language (XPath) Version 1.0, W3C Recommendation, November 1999, available via
- 104 <http://www.w3.org/TR/1999/REC-xpath-19991116>

### 105 **1.3 Non-Normative References**

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

### 107 **1.4 Conformance Targets**

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

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

121

### 122 **1.5 Overall Architecture**

123 One of the motivations of WS-HumanTask was an increasingly important need to support the ability to

124 allow any application to create human tasks in a service-oriented manner. Human tasks had traditionally

125 been created by tightly-coupled workflow management systems (WFMS). In such environments the

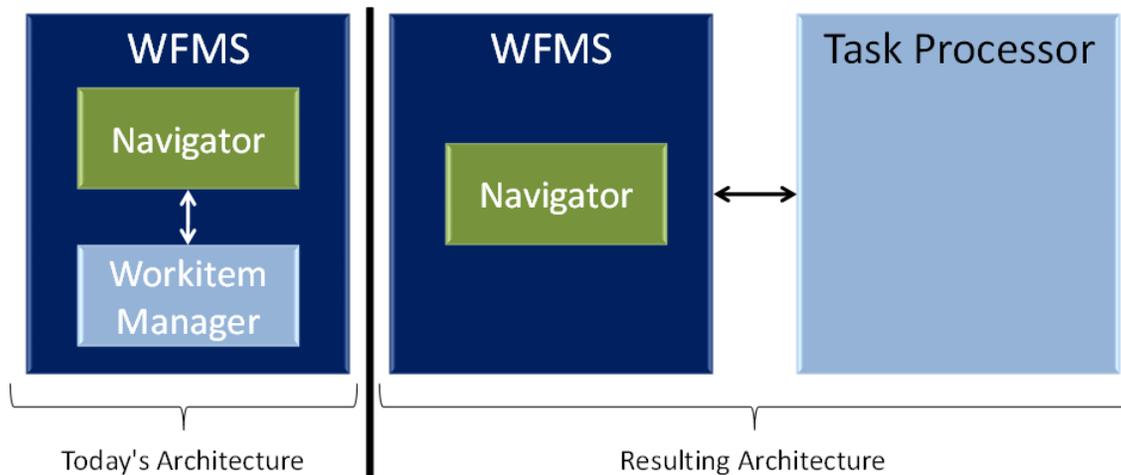
126 workflow management system managed the entirety of a task's lifecycle, an approach that did not allow

127 the means to directly affect a task's lifecycle outside of the workflow management environment (other

128 than for a human to actually carry out the task). Particularly significant was an inability to allow

129 applications to create a human task in such tightly coupled environments.

130

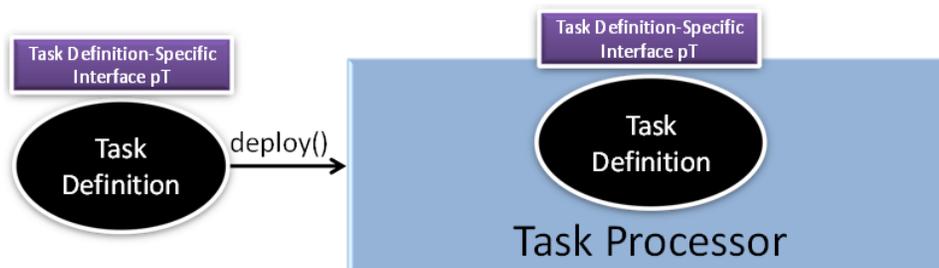


131  
132 **Figure 1- Architectural Impact of WS-HumanTask on Workflow Management Systems**

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

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

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



151  
152 **Figure 2 - Task Definitions Deployed in Task Processor**

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

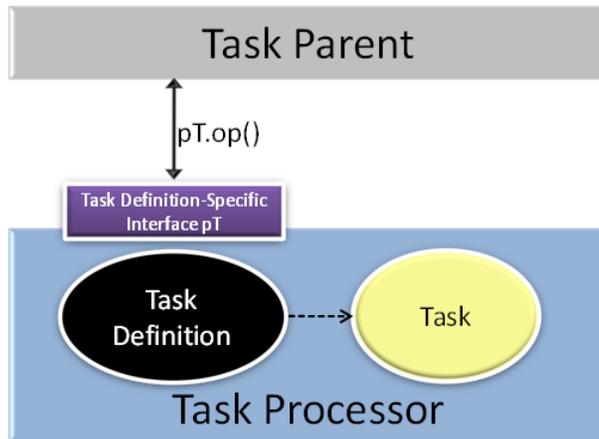


Figure 3 - Instantiating Tasks

159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179

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

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

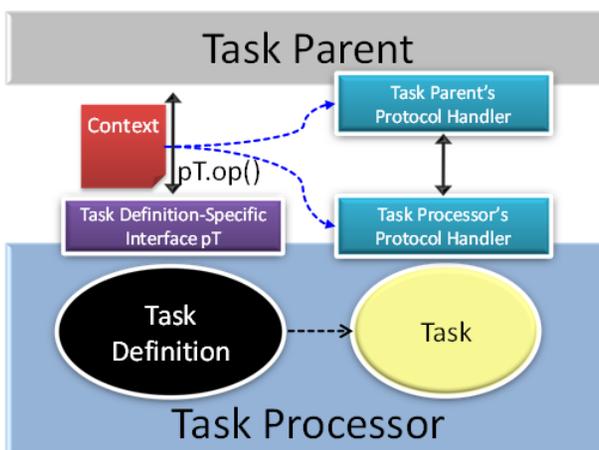
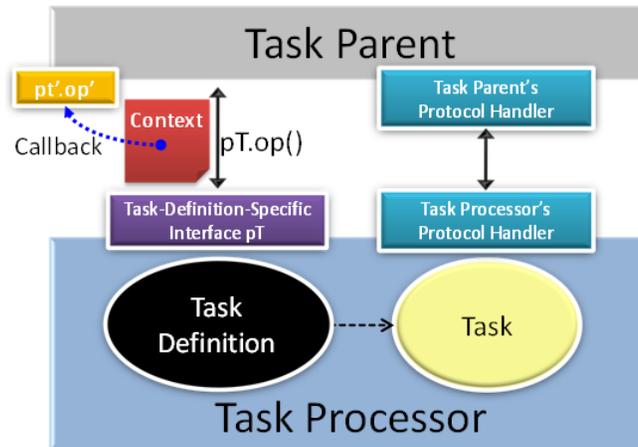


Figure 4 - Establishing a Protocol Channel

180  
181  
182  
183  
184  
185

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

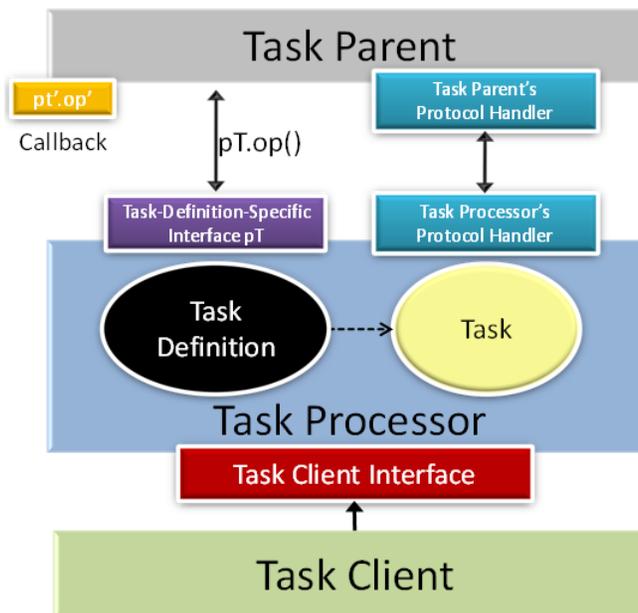
186 endpoint reference to be used to pass the result to, as well as the operation of the endpoint to be used by  
 187 the Task Processor. Figure 5 depicts this by showing that the context contains information pointing to a  
 188 port of port type *pt'* and specifying the name of the operation *op'* to be used on that port for returning  
 189 results. Note that this behavior is compliant to WS-Addressing.



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

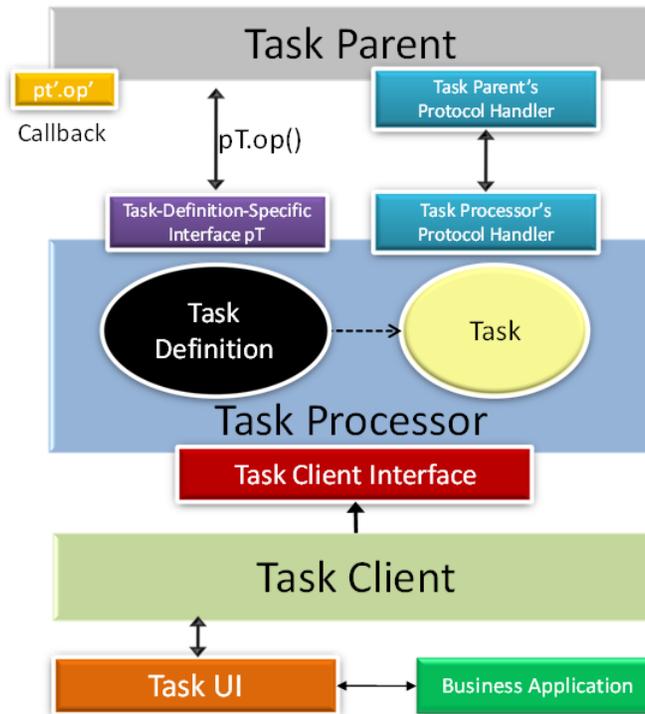
192 Finally, a Task Parent application invoking an operation implemented by a task is allowed to pass  
 193 additional data along with the request message. This data is called the *human task context* and allows the  
 194 ability to override some of the *Task Definition's* elements. Conversely, a human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

197 Once a task is created it can be presented to its (potential) owners to be claimed and worked on. For that  
 198 purpose another type of application called a *Task Client* is typically used. A Task Client presents to each  
 199 of its users the tasks available to them. Users can then decide to claim the task to carry out the work  
 200 associated with it. Other functions typically offered by a Task Client include the ability to skip a task, to  
 201 add comments or attachments to a task, to nominate other users to perform the task and that like. In  
 202 order to enable a Task Client to perform such functions on tasks, WS-HumanTask specifies the *task client*  
 203 *interface* required to be implemented by Task Processor to support Task Clients (see section 7.1). Figure  
 204 6 depicts the resultant architecture stemming from the introduction of Task Clients.



205  
 206 **Figure 6 - Task List Client and Corresponding Interface**

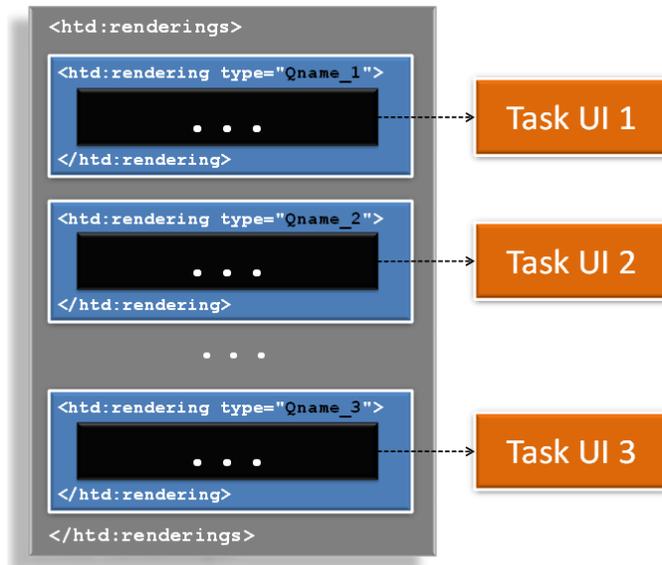
207 Once a user selects a task using his or her Task Client the user interface associated with the task is  
 208 rendered allowing the user to view application-specific information pertaining to the task. WS-HumanTask  
 209 does not specify such rendering but provides the means using a *container*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



217  
 218

Figure 7 - Overall Architecture of a Human Task Infrastructure

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



225

226

Figure 8 - Potential Renderings of a Task

227

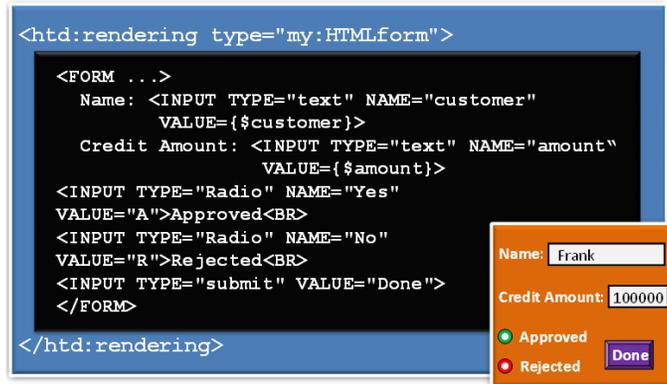
228

229

230

231

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



232

233

Figure 9 - Sample Rendering of a Task

234

235

236

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

---

## 237 2 Language Design

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

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

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

### 250 2.1 Dependencies on Other Specifications

251 WS-HumanTask utilizes the following specifications:

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

#### 258 2.1.1 Namespaces Referenced

259 WS-HumanTask references these namespaces:

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

### 264 2.2 Language Extensibility

265 The WS-HumanTask extensibility mechanism allows:

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

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

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

## 275 2.3 Overall Language Structure

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

### 279 2.3.1 Syntax

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

### 314 2.3.2 Properties

315 The <humanInteractions> element has the following properties:

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

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

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

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

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

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

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

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

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

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

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

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

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

## 404 2.4 Default use of XPath 1.0 as an Expression Language

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

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

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

---

419

## 3 Concepts

420

### 3.1 Generic Human Roles

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

- 423 • Task initiator
- 424 • Task stakeholders
- 425 • Potential owners
- 426 • Actual owner
- 427 • Excluded owners
- 428 • Business administrators
- 429 • Notification recipients

430

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

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

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

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

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

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

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

## 462 3.2 Composite Tasks and Sub Tasks

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

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

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

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

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

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

### 477 3.2.1 Composite Tasks by Definition

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

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

### 493 3.2.2 Composite Tasks Created Adhoc at Runtime

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

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

## 499 3.3 Routing Patterns

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

## 505 3.4 Relationship of Composite Tasks and Routing Patterns

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

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

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

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

## 518 3.5 Assigning People

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

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

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

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

### 531 Syntax:

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

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

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

```

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

```

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

```

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

```

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

### 582 3.5.1 Using Logical People Groups

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

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

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

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

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

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

#### 622 **Syntax:**

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

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

#### 634 **Example:**

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

### 642 **3.5.2 Using Literals**

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

646

647

648 **Syntax:**

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

654 **Example specifying user identifiers:**

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

671 **Example specifying group names:**

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

### 681 **3.5.3 Using Expressions**

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

685 **Syntax:**

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

689

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

693

694

695 **Example:**

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

706 **3.5.4 Data Type for Organizational Entities**

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

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

728 **3.5.5 Subtasks**

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

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

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

## 748 3.6 Task Rendering

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

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

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

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

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

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

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

## 778 3.7 Lean Tasks

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

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

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

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

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

## 805 **3.8 Task Instance Data**

806 Task instance data falls into three categories:

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

### 813 **3.8.1 Presentation Data**

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

### 817 **3.8.2 Context Data**

818 The task context includes the following:

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

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

### 828 **3.8.3 Operational Data**

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

### 832 3.8.3.1 Ad-hoc Attachments

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

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

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

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

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

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

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

854 The `attachedBy` element indicates who added the attachment. It ~~could be~~ is a single user, not a group or  
855 a list of users or groups. (type `htt:tUser`).

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

### 859 Attachment Info Data Type

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

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

### 875 Attachment Data Type

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

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

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

### 884 3.8.3.2 Comments

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

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

892 The `addedBy` element indicates who added the comment. It ~~could be~~ is a single user, not a group or a list  
893 of users or groups. (type `htt:tUser`).

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

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

### 897 Comment Data Type

898 The following data type is used to return comments.

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

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

### 913 3.8.4 Data Types for Task Instance Data

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

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

922 The elements `startByExists` and `completeByExists` have a value of “true” if the task has at least  
923 one start deadline or at least one completion deadline respectively. The actual times (`startByTime` and  
924 `completeByTime`) of the individual deadlines can be retrieved using the query operation (see section  
925 7.1.3 “Advanced Query Operation”).

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

927

929 **TaskAbstract Data Type**

```

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

```

984

985 **TaskDetails Data Type**

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

```

1041     <xsd:element name="hasAttachments"
1042               type="xsd:boolean" minOccurs="0"/>
1043     <xsd:element name="hasComments"
1044               type="xsd:boolean" minOccurs="0"/>
1045     <xsd:element name="escalated"
1046               type="xsd:boolean" minOccurs="0"/>
1047     <xsd:element name="searchBy"
1048               type="xsd:string" minOccurs="0"/>
1049     <xsd:element name="outcome"
1050               type="xsd:string" minOccurs="0"/>
1051     <xsd:element name="parentTaskId"
1052               type="xsd:anyURI" minOccurs="0"/>
1053     <xsd:element name="hasSubTasks"
1054               type="xsd:boolean" minOccurs="0"/>
1055     <xsd:any namespace="##other" processContents="lax"
1056            minOccurs="0" maxOccurs="unbounded"/>
1057   </xsd:sequence>
1058 </xsd:complexType>

```

### Common Data Types

```

1059 <xsd:simpleType name="tPresentationName">
1060   <xsd:annotation>
1061     <xsd:documentation>length-restricted string</xsd:documentation>
1062   </xsd:annotation>
1063   <xsd:restriction base="xsd:string">
1064     <xsd:maxLength value="64" />
1065     <xsd:whiteSpace value="preserve" />
1066   </xsd:restriction>
1067 </xsd:simpleType>
1068
1069 <xsd:simpleType name="tPresentationSubject">
1070   <xsd:annotation>
1071     <xsd:documentation>length-restricted string</xsd:documentation>
1072   </xsd:annotation>
1073   <xsd:restriction base="xsd:string">
1074     <xsd:maxLength value="254" />
1075     <xsd:whiteSpace value="preserve" />
1076   </xsd:restriction>
1077 </xsd:simpleType>
1078
1079 <xsd:simpleType name="tStatus">
1080   <xsd:restriction base="xsd:string" />
1081 </xsd:simpleType>
1082
1083 <xsd:simpleType name="tPredefinedStatus">
1084   <xsd:annotation>
1085     <xsd:documentation>for documentation only</xsd:documentation>
1086   </xsd:annotation>
1087   <xsd:restriction base="xsd:string">
1088     <xsd:enumeration value="CREATED" />
1089     <xsd:enumeration value="READY" />
1090     <xsd:enumeration value="RESERVED" />
1091     <xsd:enumeration value="IN_PROGRESS" />
1092     <xsd:enumeration value="SUSPENDED" />
1093     <xsd:enumeration value="COMPLETED" />
1094     <xsd:enumeration value="FAILED" />
1095     <xsd:enumeration value="ERROR" />
1096     <xsd:enumeration value="EXITED" />
1097   </xsd:restriction>

```

```
1098 <xsd:enumeration value="OBSOLETE" />
1099 </xsd:restriction>
1100 </xsd:simpleType>
```

### 1101 3.8.5 Sub Tasks

1102 To support sub tasks the task instance data gets enhanced by the following (optional) parameters:

- 1103 • sub tasks → A list of task identifiers for each already-created subtask of the task, including
- 1104 both non-terminated and terminated instances
- 1105 → A list of the names of the sub tasks available for creation in the definition of the
- 1106 task, based on the composition type, instantiation pattern, and already created tasks
- 1107 • parent task → The identifier of the superior composite task of this task if it is a sub task

---

## 1108 4 Human Tasks

1109 The <task> element is used to specify human tasks. This section introduces the syntax for the element,  
1110 and individual properties are explained in subsequent sections.

### 1111 4.1 Overall Syntax

1112 Definition of human tasks:

```
1113 <htd:task name="NCName" actualOwnerRequired="yes|no"?>
1114
1115   <htd:interface portType="QName" operation="NCName"
1116     responsePortType="QName"? responseOperation="NCName"? />
1117
1118   <htd:priority expressionLanguage="anyURI"? >?
1119     integer-expression
1120 </htd:priority>
1121
1122   <htd:peopleAssignments>?
1123     ...
1124 </htd:peopleAssignments>
1125
1126   <htd:completionBehavior>?
1127     ...
1128 </htd:completionBehavior>
1129
1130   <htd:delegation
1131     potentialDelegates="anybody|nobody|potentialOwners|other"/>?>?
1132     <htd:from>?
1133       ...
1134     </htd:from>
1135 </htd:delegation>
1136
1137   <htd:presentationElements>?
1138     ...
1139 </htd:presentationElements>
1140
1141   <htd:possibleOutcomes>?
1142     ...
1143 </htd:possibleOutcomes>
1144
1145   <htd:outcome part="NCName" queryLanguage="anyURI">?
1146     queryContent
1147 </htd:outcome>
1148
1149   <htd:searchBy expressionLanguage="anyURI"? >?
1150     expression
1151 </htd:searchBy>
1152
1153   <htd:renderings>?
1154     <htd:rendering type="QName">+
1155       ...
1156     </htd:rendering>
1157 </htd:renderings>
1158
1159   <htd:deadlines>?
```

```

1160
1161     <htd:startDeadline name="NCName"*
1162     ...
1163     </htd:startDeadline>
1164
1165     <htd:completionDeadline name="NCName"*
1166     ...
1167     </htd:completionDeadline>
1168
1169 </htd:deadlines>
1170
1171 <htd:composition>?
1172     ...
1173 </htd:composition>
1174
1175 </htd:task>

```

## 1176 4.2 Properties

1177 The following attributes and elements are defined for tasks:

- 1178 • **name**: This attribute is used to specify the name of the task. The name combined with the target  
1179 namespace MUST uniquely identify a task element enclosed in the task definition. This attribute  
1180 is mandatory. It is not used for task rendering.
- 1181 • **actualOwnerRequired**: This optional attribute specifies if an actual owner is required for the  
1182 task. Setting the value to "no" is used for composite tasks where subtasks should be activated  
1183 automatically without user interaction. For routing tasks [h:isThis](#) attribute MUST be set to "no".  
1184 Tasks that have been defined to not have subtasks MUST have exactly one actual owner after  
1185 they have been claimed. For these tasks the value of the attribute value MUST be "yes". The  
1186 default value for the attribute is "yes".
- 1187 • **interface**: This element is used to specify the operation used to invoke the task. The operation  
1188 is specified using WSDL, that is, a WSDL port type and WSDL operation are defined. The  
1189 element and its `portType` and `operation` attributes MUST be present for normal tasks. The  
1190 schema only marks it optional so that Lean Tasks can make it prohibited. The interface is  
1191 specified in one of the following forms:
  - 1192 ▪ The WSDL operation is a **one-way** operation and the task asynchronously  
1193 returns output data. In this case, a WS-HumanTask Definition MUST specify a  
1194 callback one-way operation, using the `responsePortType` and  
1195 `responseOperation` attributes. This callback operation is invoked when the  
1196 task has finished. The Web service endpoint address of the callback operation is  
1197 provided at runtime when the task's one-way operation is invoked (for details,  
1198 see section 10 "Providing Callback Information for Human Tasks").
  - 1199 ▪ The WSDL operation is a **request-response** operation. In this case, the  
1200 `responsePortType` and `responseOperation` attributes MUST NOT be  
1201 specified.
- 1202 • **priority**: This element is used to specify the priority of the task. It is an optional element which  
1203 value is an integer expression. If present, the WS-HumanTask Definition MUST specify a value  
1204 between 0 and 10, where 0 is the highest priority and 10 is the lowest. If not present, the priority  
1205 of the task is considered as 5. The result of the expression evaluation is of type  
1206 `http:tPriority`. The `expressionLanguage` attribute specifies the language used in the  
1207 expression. The attribute is optional. If not specified, the default language as inherited from the  
1208 closest enclosing element that specifies the attribute is used.

- 1209 • `peopleAssignments`: This element is used to specify people assigned to different generic  
1210 human roles, i.e. potential owners, and business administrator. The element is optional. See  
1211 section 3.5 for more details on people assignments.
- 1212 • `completionBehavior`: This element is used to specify completion conditions of the task. It is  
1213 optional. See section 4.8 for more details on completion behavior.
- 1214 • `delegation`: This element is used to specify constraints concerning delegation of the task.  
1215 Attribute `potentialDelegates` defines to whom the task can be delegated. One of the  
1216 following values MUST be specified:
  - 1217 ▪ `anybody`: It is allowed to delegate the task to anybody
  - 1218 ▪ `potentialOwners`: It is allowed to delegate the task to potential owners  
1219 previously selected
  - 1220 ▪ `other`: It is allowed to delegate the task to other people, e.g. authorized owners.  
1221 The element `<from>` is used to determine the people to whom the task can be  
1222 delegated.
  - 1223 ▪ `nobody`: It is not allowed to delegate the task.
- 1224 The delegation element is optional. If this element is not present the task is allowed to be  
1225 delegated to anybody.
- 1226 • `presentationElements`: This element is used to specify different information used to display  
1227 the task in a task list, such as name, subject and description. See section 4.3 for more details on  
1228 presentation elements. The element is optional.
- 1229 • `outcome`: This optional element identifies the field (of an xsd simple type) in the output message  
1230 which reflects the business result of the task. A conversion takes place to yield an outcome of  
1231 type `xsd:string`. The optional attribute `queryLanguage` specifies the language used for  
1232 selection. If not specified, the default language as inherited from the closest enclosing element  
1233 that specifies the attribute is used.
- 1234 • `searchBy`: This optional element is used to search for task instances based on a custom search  
1235 criterion. The result of the expression evaluation is of type `xsd:string`. The  
1236 `expressionLanguage` attribute specifies the language used in the expression. The attribute is  
1237 optional. If not specified, the default language as inherited from the closest enclosing element that  
1238 specifies the attribute is used.
- 1239 • `rendering`: This element is used to specify the rendering method. It is optional. If not present,  
1240 task rendering is implementation dependent. See section 4.4 for more details on rendering tasks.
- 1241 • `deadlines`: This element specifies different deadlines. It is optional. See section 4.9 for more  
1242 details on timeouts and escalations.
- 1243 • `composition`: This element is used to specify subtasks of a composite task. It is optional. See  
1244 section 4.6 for more details on composite tasks.

## 1245 4.3 Presentation Elements

1246 Information about human tasks or notifications needs to be made available in a human-readable way to  
1247 allow users dealing with their tasks and notifications via a user interface, which could be based on various  
1248 technologies, such as Web browsers, Java clients, Flex-based clients or .NET clients. For example, a  
1249 user queries for her tasks, getting a list of tasks she could work on, displaying a short description of each  
1250 task. Upon selection of one of the tasks, more complete information about the task is displayed by the  
1251 user interface.

1252 Alternatively, a task or notification could be sent directly to a user's inbox, in which case the same  
1253 information would be used to provide a human readable rendering there.

1254 The same human readable information could also be used in reports on all the human tasks executed by  
1255 a particular human task management system.

1256 Human readable information can be specified in multiple languages.

### 1257 **Syntax:**

```
1258 <htd:presentationElements>
1259
1260   <htd:name xml:lang="xsd:language"? >*
1261     Text
1262   </htd:name>
1263
1264   <!-- For the subject and description only,
1265     replacement variables can be used. -->
1266   <htd:presentationParameters expressionLanguage="anyURI"? >?
1267     <htd:presentationParameter name="NCName" type="QName">+
1268       expression
1269     </htd:presentationParameter>
1270   </htd:presentationParameters>
1271
1272   <htd:subject xml:lang="xsd:language"? >*
1273     Text
1274   </htd:subject>
1275
1276   <htd:description xml:lang="xsd:language"?
1277     contentType="mimeTypeString"? >*
1278     <xsd:any minOccurs="0" maxOccurs="unbounded" />
1279   </htd:description>
1280
1281 </htd:presentationElements>
```

### 1282 **Properties**

1283 The following attributes and elements are defined for the `htd:presentationElements` element.

- 1284 • `name`: This element is the short title of a task. It uses `xml:lang`, a standard XML attribute, to  
1285 define the language of the enclosed information. This attribute uses tags according to RFC 1766  
1286 (see [RFC1766]). There could be zero or more `name` elements. A WS-HumanTask Definition  
1287 MUST NOT specify multiple `name` elements having the same value for attribute `xml:lang`.
- 1288 • `presentationParameters`: This element specifies parameters used in presentation elements  
1289 `subject` and `description`. Attribute `expressionLanguage` identifies the expression  
1290 language used to define parameters. This attribute is optional. If not specified, the default  
1291 language as inherited from the closest enclosing element that specifies the attribute is used.  
1292 Element `presentationParameters` is optional and if present then the WS-HumanTask  
1293 Definition MUST specify at least one element `presentationParameter`. Element  
1294 `presentationParameter` has attribute `name`, which uniquely identifies the parameter  
1295 definition within the `presentationParameters` element, and attribute `type` which defines its  
1296 type. A WS-HumanTask Definition MUST specify parameters of XSD simple types. When a  
1297 `presentationParameter` is used within `subject` and `description`, the syntax is  
1298 `{${parameterName}}`. The pair `"{"` represents the character `"{"` and the pair `"}"` represents  
1299 the character `"}"`. Only the defined presentation parameters are allowed, that is, a WS-  
1300 HumanTask Definition MUST NOT specify arbitrary expressions embedded in this syntax.
- 1301 • `subject`: This element is a longer text that describes the task. It uses `xml:lang` to define the  
1302 language of the enclosed information. There could be zero or more `subject` elements. A WS-  
1303 HumanTask Definition MUST NOT specify multiple `subject` elements having the same value for  
1304 attribute `xml:lang`.
- 1305 • `description`: This element is a long description of the task. It uses `xml:lang` to define the  
1306 language of the enclosed information. The optional attribute `contentType` uses content types

1307 according to RFC 2046 (see [RFC 2046]). The default value for this attribute is "text/plain". A WS-  
1308 HumanTask Processor MUST support the content type "text/plain". The WS-HumanTask  
1309 Processor SHOULD support HTML (such as "text/html" or "application/xml+xhtml"). There could  
1310 be zero or more description elements. As descriptions can exist with different content types, it  
1311 is allowed to specify multiple description elements having the same value for attribute  
1312 xml:lang, but the WS-HumanTask Definition MUST specify different content types.

1313 **Example:**

```
1314 <htd:presentationElements>
1315
1316   <htd:name xml:lang="en-US">Approve Claim</htd:name>
1317   <htd:name xml:lang="de-DE">
1318     Genehmigung der Schadensforderung
1319   </htd:name>
1320
1321   <htd:presentationParameters>
1322     <htd:presentationParameter name="firstname" type="xsd:string">
1323       htd:getInput("ClaimApprovalRequest")/cust/firstname
1324     </htd:presentationParameter>
1325     <htd:presentationParameter name="lastname" type="xsd:string">
1326       htd:getInput("ClaimApprovalRequest")/cust/lastname
1327     </htd:presentationParameter>
1328     <htd:presentationParameter name="euroAmount" type="xsd:double">
1329       htd:getInput("ClaimApprovalRequest")/amount
1330     </htd:presentationParameter>
1331   </htd:presentationParameters>
1332
1333   <htd:subject xml:lang="en-US">
1334     Approve the insurance claim for €{$euroAmount} on behalf of
1335     {$firstname} {$lastname}
1336   </htd:subject>
1337   <htd:subject xml:lang="de-DE">
1338     Genehmigung der Schadensforderung über €{$euroAmount} für
1339     {$firstname} {$lastname}
1340   </htd:subject>
1341
1342   <htd:description xml:lang="en-US" contentType="text/plain">
1343     Approve this claim following corporate guideline #4711.0815/7 ...
1344   </htd:description>
1345   <htd:description xml:lang="en-US" contentType="text/html">
1346     <p>
1347       Approve this claim following corporate guideline
1348       <b>#4711.0815/7</b>
1349       ...
1350     </p>
1351   </htd:description>
1352   <htd:description xml:lang="de-DE" contentType="text/plain">
1353     Genehmigen Sie diese Schadensforderung entsprechend Richtlinie Nr.
1354     4711.0815/7 ...
1355   </htd:description>
1356   <htd:description xml:lang="de-DE" contentType="text/html">
1357     <p>
1358       Genehmigen Sie diese Schadensforderung entsprechend Richtlinie
1359       <b>Nr. 4711.0815/7</b>
1360       ...
1361     </p>
1362   </htd:description>
```

1363  
1364 </htd:presentationElements>

## 1365 4.4 Task Possible Outcomes

1366 The <possibleOutcomes> element provides a way for a task to define which values are usable for the  
1367 outcome value of a task. Having a separate definition allows a tool for building tasks to provide support  
1368 that understands exactly which outcomes are possible for a particular task.

```
1369 <htd:possibleOutcomes>  
1370   <htd:possibleOutcome name="NCName">+  
1371     <htd:outcomeName xml:lang="xsd:language"?>+  
1372       Language specific display  
1373     </htd:outcomeName>  
1374   </htd:possibleOutcome>  
1375 </htd:possibleOutcomes>
```

1376 Each <possibleOutcome> element represents one possible outcome. For the typical example of an  
1377 expense report approval, the two outcomes might be 'Approve' and 'Reject'. In addition to the other data  
1378 being collected by the rendering in the WS-HumanTask Client, this represents the most important  
1379 information about how to proceed in a process that contains multiple tasks. Therefore, a rendering and  
1380 client using HTML might choose to show this as a dropdown list, list box with single selection, a set of  
1381 submit buttons, or a radio button group.

1382 For each <possibleOutcome>, it is possible to have an <outcomeName> element to specify a per-  
1383 language display name. It uses `xml:lang`, a standard XML attribute, to define the language of the  
1384 enclosed information. This attribute uses tags according to RFC 1766 (see [RFC1766]). There could be  
1385 zero or more <outcomeName> elements. A <possibleOutcome> MUST NOT specify multiple  
1386 <outcomeName> elements having the same value for attribute `xml:lang`.

## 1387 4.5 Elements for Rendering Tasks

1388 Human tasks and notifications need to be rendered on user interfaces like forms clients, portlets, e-mail  
1389 clients, etc. The rendering element provides an extensible mechanism for specifying UI renderings for  
1390 human tasks and notifications (task-UI). The element is optional. One or more rendering methods can be  
1391 provided in a task definition or a notification definition. A task or notification can be deployed on any WS-  
1392 HumanTask Processor, irrespective of the fact whether the implementation supports specified rendering  
1393 methods or not. The rendering method is identified using a QName.

1394 Unlike for presentation elements, language considerations are opaque for the rendering element because  
1395 the rendering applications typically provide multi-language support. Where this is not the case, providers  
1396 of certain rendering types can decide to extend the rendering method in order to provide language  
1397 information for a given rendering.

1398 The content of the rendering element is not defined by this specification. For example, when used in the  
1399 rendering element, XPath extension functions as defined in section 7.2 MAY be evaluated by a WS-  
1400 HumanTask Processor.

1401

1402

1403 **Syntax:**

```
1404 <htd:renderings>
1405   <htd:rendering type="QName">+
1406     <xsd:any minOccurs="1" maxOccurs="1" />
1407   </htd:rendering>
1408 </htd:renderings>
```

1409 **4.6 Elements for Composite Tasks**

1410 A composite task is defined as a `<htd:task>` element with the `<htd:composition>` element enclosed  
1411 in it. The following are attributes and elements defined for the `composition` element.

- 1412 • `type`: This optional attribute specifies the order in which enclosed sub-tasks are executed. If the  
1413 value is set to “sequential” the sub-tasks MUST be executed in lexical order. Otherwise they  
1414 MUST be executed in parallel. The default value for this attribute is “sequential”.
- 1415 • `instantiationPattern`: This optional attribute specifies the way `how` sub-tasks are  
1416 instantiated. If the value is set to “manual” the task client triggers instantiation of enclosed sub-  
1417 tasks. Otherwise, they are automatically instantiated at the time the composite task itself turns  
1418 into status “inProgress”. The default value for this attribute is “manual”.
- 1419 • `subtask`: This element specifies a task that will be executed as part of the composite task  
1420 execution. The `composition` element MUST enclose at least one `subtask` element. The  
1421 `subtask` element has the following attributes and elements. The `name` attribute specifies the  
1422 name of the sub-task. The name MUST be unique among the names of all sub-tasks within the  
1423 `composition` element. The `htd:task` element is used to define the task inline. The  
1424 `htd:localTask` element is used to reference a task that will be executed as a sub-task. The  
1425 `htd:localTask` element MAY define values for standard overriding attributes: priority and  
1426 people assignments. The `toParts` element is used to assign values to input message of the  
1427 sub-task. The enclosed XPath expression MAY refer to the input message of the composite task  
1428 or the output message of other sub-task enclosed in the same `composition` element. The  
1429 `part` attribute refers to a part of the WSDL message type of the message used in the XPath.  
1430 The `expressionLanguage` attribute specifies the expression language used in the enclosing  
1431 elements. The default value for this attribute is `urn:ws-ht:sublang:xpath1.0` which  
1432 represents the usage of XPath 1.0 within human interactions definition. A WS-HumanTask  
1433 Definition that uses expressions MAY override the default expression language for individual  
1434 expressions.

1435 When `composition` is defined on a task, the `composition` MUST be applied for each of the potential  
1436 owners defined in the task’s people assignment.

1437 **Syntax:**

```
1438 <htd:task>
1439   ...
1440   <htd:composition type="sequential|parallel"
1441     instantiationPattern="manual|automatic">
1442     <htd:subtask name="NCName">+
1443       (
1444         <htd:task>
1445           ...
1446         </htd:task>
1447       |
1448         <htd:localTask reference="QName">
1449           standard-overriding-elements
1450           ...
1451         </htd:localTask>
```

```

1452     )
1453 |
1454     <htd:toParts>?
1455     <htd:toPart part="NCName" expressionLanguage="anyURI">+
1456     XPath expression
1457     </htd:toPart>
1458     </htd:toParts>
1459 |
1460     </htd:subtask>
1461 |
1462     </htd:composition>
1463     ...
1464 </htd:task>

```

1465 *Standard-overriding-elements* is used in the syntax above as a shortened form of the following list of  
1466 elements:

```

1467 <htd:priority expressionLanguage="anyURI"? >
1468     integer-expression
1469 </htd:priority>
1470
1471 <htd:peopleAssignments>?
1472     <htd:genericHumanRole>
1473     <htd:from>...</htd:from>
1474     </htd:genericHumanRole>
1475 </htd:peopleAssignments>

```

## 1476 4.7 Elements for People Assignment

1477 The <peopleAssignments> element is used to assign people to a task. For each generic human role, a  
1478 people assignment element can be specified. A WS-HumanTask Definition MUST specify a people  
1479 assignment for potential owners of a human task. An empty <potentialOwners> element is used to  
1480 specify that no potential owner is assigned by the human task's definition but another means is used e.g.  
1481 nomination. Specifying people assignments for task stakeholders, task initiators, excluded owners and  
1482 business administrators is optional. Human tasks never specify recipients. A WS-HumanTask Definition  
1483 MUST NOT specify people assignments for actual owners.

### 1484 Syntax:

```

1485 <htd:peopleAssignments>
1486 |
1487     <htd:potentialOwners>
1488     ...
1489     </htd:potentialOwners>
1490 |
1491     <htd:excludedOwners>?
1492     ...
1493     </htd:excludedOwners>
1494 |
1495     <htd:taskInitiator>?
1496     ...
1497     </htd:taskInitiator>
1498 |
1499     <htd:taskStakeholders>?
1500     ...
1501     </htd:taskStakeholders>
1502 |
1503     <htd:businessAdministrators>?
1504     ...
1505     </htd:businessAdministrators>

```

1506 |  
1507 | `</htd:peopleAssignments>`

1508 | People assignments can result in a set of values or an empty set. In case people assignment results in an  
1509 | empty set then the task potentially requires administrative attention. This is out of scope of the  
1510 | specification, except for people assignments for potential owners (see section 4.10.1 “Normal processing  
1511 | of a Human Task” for more details).

1512 |  
1513 |

#### 1514 | **Example:**

```
1515 | <htd:peopleAssignments>  
1516 |   <htd:potentialOwners>  
1517 |     <htd:from logicalPeopleGroup="regionalClerks">  
1518 |       <htd:argument name="region">  
1519 |         htd:getInput("ClaimApprovalRequest")/region  
1520 |       </htd:argument>  
1521 |     </htd:from>  
1522 |   </htd:potentialOwners>  
1523 |  
1524 |   <htd:businessAdministrators>  
1525 |     <htd:from logicalPeopleGroup="regionalManager">  
1526 |       <htd:argument name="region">  
1527 |         htd:getInput("ClaimApprovalRequest")/region  
1528 |       </htd:argument>  
1529 |     </htd:from>  
1530 |   </htd:businessAdministrators>  
1531 | </htd:peopleAssignments>
```

## 1532 | **4.7.1 Routing Patterns**

1533 | Tasks can be assigned to people in sequence and parallel. Elements `htd:sequence` and  
1534 | `htd:parallel` elements in `htd:potentialOwners` are used to represent such assignments.

### 1535 | **4.7.1.1 Parallel Pattern**

1536 | A task can be assigned to people in parallel using the `htd:parallel` element. The `htd:parallel`  
1537 | element is defined as follows:

- 1538 | • The `htd:from` element defines the parallel potential owners. This can evaluate to multiple  
1539 | users/groups.
- 1540 | • The attribute ‘type’ in `htd:parallel` identifies how parallel assignments are created for the  
1541 | multiple users/groups returned from `htd:from`. If type is ‘all’ then an assignment MUST be  
1542 | created for each user returned by `htd:from`. If type is ‘single’ then an assignment MUST be  
1543 | created for each `htd:from` clause (this assignment could have `with-n` potential owners). The  
1544 | default value of type is ‘all’.
- 1545 | • The `htd:parallel` and `htd:sequence` elements define nested routing patterns within the  
1546 | parallel routing pattern.
- 1547 | • The `htd:completionBehavior` defines when the routing pattern completes. The completion  
1548 | criteria also define how the result is constructed for the parent task when a parallel routing  
1549 | pattern is complete.

1550 | Each parallel assignment MUST result in a separate sub task. Sub tasks created for each parallel  
1551 | assignment MUST identify the parent task using the `htd:parentTaskId`.

1552 |

1553

1554 **Syntax:**

```
1555 <htd:potentialOwners>  
1556   <htd:parallel type="all|single"?>  
1557     <htd:completionBehavior>>?  
1558     ...  
1559   </htd:completionBehavior>  
1560   <htd:from>...</>*</>  
1561   ...  
1562 </htd:parallel>  
1563   pattern*  
1564 </htd:parallel>  
1565 </htd:potentialOwners>
```

1566 **Example:**

```
1567 <htd:peopleAssignments>  
1568   <htd:potentialOwners>  
1569     <htd:parallel type="all">  
1570       <htd:from>  
1571         htd:getInput("ClaimApprovalRequest")/claimAgent  
1572       </htd:from>  
1573     </htd:parallel>  
1574   </htd:potentialOwners>  
1575 </htd:peopleAssignments>
```

1576 **4.7.1.2 Sequential Pattern**

1577 A task can be assigned to people in sequence using the `htd:sequence` element. The `htd:sequence`  
1578 is defined as follows:

- 1579 • The `htd:from` element can evaluate to multiple users/groups.
- 1580 • The attribute 'type' in `htd:sequence` identifies how sequential assignments are created for the  
1581 multiple users/groups returned from `htd:from`. If type is 'all' an assignment MUST be created  
1582 for each user returned by `htd:from`. If type is 'single', an assignment MUST be created for each  
1583 `htd:from` clause (this assignment could have with n potential owners). The default value of type  
1584 is 'all'.
- 1585 • The `htd:parallel` and `htd:sequence` elements define nested routing patterns within the  
1586 sequential routing pattern.
- 1587 • The `htd:completionBehavior` defines when the routing pattern completes. The completion  
1588 criteria also define how the result is constructed for the parent task when a sequential routing  
1589 pattern is complete.

1590 Sequential routing patterns MUST use a separate sub task for each step in a sequential pattern. Sub  
1591 tasks created for each sequential assignment MUST identify the parent task using the  
1592 `htd:parentTaskId`.

1593

1594 |

1595 **Syntax:**

```
1596 <htd:potentialOwners>  
1597   <htd:sequence type="all|single"?>  
1598     <htd:completionBehavior?>  
1599       ...  
1600     </htd:completionBehavior>  
1601     <htd:from>...</>*<u>  
1602       ...  
1603     </htd:from>*<u>  
1604     pattern*  
1605   </htd:sequence>  
1606 </htd:potentialOwners>
```

1607 |

1608 **Example:**

```
1609 <htd:peopleAssignments>
1610   <htd:potentialOwners>
1611     <htd:sequence type="all">
1612       <htd:from logicalPeopleGroup="regionalClerks">
1613         <htd:argument name="region">
1614           htd:getInput("ClaimApprovalRequest")/region
1615         </htd:argument>
1616       </htd:from>
1617       <htd:from logicalPeopleGroup="regionalManager">
1618         <htd:argument name="region">
1619           htd:getInput("ClaimApprovalRequest")/region
1620         </htd:argument>
1621       </htd:from>
1622     </htd:sequence>
1623   </htd:potentialOwners>
1624 </htd:peopleAssignments/>
```

1625 **4.8 Completion Behavior**

1626 The completion behavior of a task, routing pattern or composite task can be influenced by a specification  
1627 of completion conditions and the result construction for tasks, routing patterns, or composite tasks. For  
1628 this purpose, the task, routing pattern or composite task contains a `htd:completionBehavior`  
1629 element.

1630 Multiple completion conditions can be specified as nested `htd:completion` elements. They are  
1631 evaluated in lexical order. When one of the specified completion conditions is met then the task is  
1632 considered to be completed; in case of routing patterns and composite tasks all remaining running sub  
1633 tasks **MUST** be skipped (i.e., set to the "Obsolete" state) and the associated result construction **MUST** be  
1634 applied.

1635 In case of composite tasks and routing patterns the following applies: At most one default completion  
1636 **MUST** be specified with no completion condition in order to specify the result construction after regular  
1637 completion of all sub tasks. If no result construction is applied, e.g. because no "default" result  
1638 construction is specified and none of the specified completion conditions is met, then the parent task's  
1639 output is not created, i.e., it remains uninitialized. Moreover, note that a completion condition can be  
1640 specified without referencing sub task output data, which allows the parent task to be considered  
1641 completed even without creating any sub tasks. When output data from sub tasks is referenced by  
1642 completion conditions or result constructions, only output data of already finished sub tasks **MUST** be  
1643 considered.

1644 If none of the specified completion conditions is met then the state of the task or the parent task remains  
1645 unchanged.

```
1646 |
1647 | <htd:completionBehavior completionAction="="manual|automatic"???"???"???">
1648 |   <htd:completion name="NCName">*
1649 |     <htd:condition ... >
1650 |       ...
1651 |     </htd:condition>
1652 |     <htd:result>?
1653 |     ...
1654 |     <htd:result>
1655 |   </htd:completion>
1656 |   <htd:defaultCompletion>?
1657 |     <htd:result>
1658 |     ...
1659 |     <htd:result>
1660 |   </htd:defaultCompletion>
```

1661 `</htd:completionBehavior>`

1662 The `completionBehavior` element has optional attribute `completionAction`. This optional  
1663 attribute specifies how the task, routing pattern, or composite task is completed. If the value is set to  
1664 "manual" the task or parent task MUST be completed explicitly by the actual owner as soon as the  
1665 completion conditions evaluate to true. If the value is set to "automatic" the task or parent task MUST be  
1666 set to complete as soon as the completion conditions evaluate to true. For routing patterns, the  
1667 `completionAction` attribute MUST have value "automatic". The default value for this attribute is  
1668 "automatic".

1669 If `completionBehavior` is not specified, the default behavior is that of a `completionBehavior` with  
1670 `completionCondition` is "true" and a `completionAction` of "manual" for simple and composite  
1671 tasks, and "automatic" for routing patterns.

## 1672 4.8.1 Completion Conditions

1673 A completion condition defines when a task or a set of sub tasks associated with the parent task is  
1674 considered completed. It is specified Boolean expression which can refer to input data of the task, the  
1675 parent task or its sub tasks, output data produced by already finished sub tasks, or other data obtained  
1676 from WS-HumanTask API calls (e.g. the number of sub tasks), or functions that test that some designated  
1677 amount of time has passed.

1678 The completion condition MUST be defined using an `htd:condition` element.

```
1679 <htd:condition expressionLanguage="anyURI"?>  
1680   boolean expression  
1681 </htd:condition>
```

1682 Within the Boolean expression of a completion condition, aggregation functions can be used to evaluate  
1683 output data produced by the already finished sub tasks of the parent task.

1684 If an error (e.g. division by zero) occurs during the condition evaluation then the condition MUST be  
1685 considered to have evaluated to "false".

1686 The time functions that are available are defined as follows:

- 1687 • `boolean htd:waitFor(string)`
  - 1688 ○ The parameter is an XPath expression evaluating to a string conforming to the definition
  - 1689 of the XML Schema type `duration`
  - 1690 ○ The return value is `true` after the specified duration has elapsed, otherwise `false`
- 1691 • `boolean htd:waitUntil(string)`
  - 1692 ○ The parameter is an XPath expression evaluating to a string conforming to the definition
  - 1693 of the XML Schema type `dateTime`
  - 1694 ○ The return value is `true` after the specified absolute time has passed, otherwise `false`.

1695 | Completion conditions of a task [without subtasks](#) MUST use only time functions.

### 1696 4.8.1.1 Evaluating the Completion Condition

1697 The time functions in the completion condition are be evaluated with respect to the beginning of execution  
1698 of the task or parent task on which the completion is defined. To achieve this, the evaluation of the  
1699 `htd:waitFor` and `htd:waitUntil` calls within the condition are treated differently from the rest of the  
1700 expression. When the containing task or parent task is created, the actual parameter expression for any  
1701 `htd:waitFor` and `htd:waitUntil` calls MUST be evaluated and the completion condition should be  
1702 rewritten to replace these calls with only `htd:waitUntil` calls with constant parameters. The durations  
1703 calculated for any `htd:waitFor` calls MUST be converted into absolute times and rewritten as  
1704 `htd:waitUntil` calls. The result of these replacements is called the *preprocessed completion*  
1705 *condition*.

1706 |

1707  
1708 For the parent task, the preprocessed completion condition MUST be evaluated at the following times:

- 1709 • Before starting the first subtask (it may be complete before it starts)
- 1710 • Whenever a subtask completes
- 1711 • Whenever a duration specified in a `htd:waitFor` call has elapsed
- 1712 • Whenever an absolute time specified in a `htd:waitUntil` call is passed.

1713 For tasks, the preprocessed completion condition MUST be evaluated at the following times:

- 1714 • Before starting the task (it may be complete before it starts)
- 1715 • Whenever a duration specified in a `htd:waitFor` call has elapsed
- 1716 • Whenever an absolute time specified in a `htd:waitUntil` call is passed.

1717 **Example:**

1718 The first completion condition may be met even without starting sub tasks. When both parts of the second  
1719 completion condition are met, that is, 7 days have expired and more than half of the finished sub tasks  
1720 have an outcome of “Rejected”, then the parallel routing pattern is considered completed.

```
1721 <htd:parallel>  
1722   ...  
1723   <htd:completionBehavior>  
1724     <htd:completion>  
1725       <htd:condition>  
1726         htd:getInput("ClaimApprovalRequest")/amount < 1000  
1727       </htd:condition>  
1728       <htd:result> ... </htd:result>  
1729     </htd:completion>  
1730     <htd:completion>  
1731       <htd:condition>  
1732         ( htd:getCountOfSubtasksWithOutcome("Rejected") /  
1733           htd:getCountOfSubtasks() > 0.5 )  
1734         and htd:waitFor("P7D")  
1735       </htd:condition>  
1736       <htd:result> ... </htd:result>  
1737     </htd:completion>  
1738   </htd:completionBehavior>  
1739   ...  
1740 </htd:parallel>
```

1741

1742 **4.8.2 Result Construction from Parallel Subtasks**

1743 When multiple sub tasks are created in order let several people work on their own sub task in parallel  
1744 then the outputs of these sub tasks sometimes need to be combined for the creation of the parent task's  
1745 output.

1746 If all sub tasks have the same interface definition (as in routing patterns) then the result construction can  
1747 be defined in a declarative way using aggregation functions. Alternatively, the result may be created using  
1748 explicit assignments.

1749 The result construction MUST be defined as `htd:result` element, containing one or more  
1750 `htd:aggregate` or `htd:copy` elements, executed in the order in which they appear in the task  
1751 definition.

```
1752 <htd:result>  
1753   (  
1754     <htd:aggregate ... />
```

```
1755 |
1756 |     <htd:copy> ... </htd:copy>
1757 | )+
1758 </htd:result>
```

#### 4.8.2.1 Declarative Result Aggregation

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

```
1764 <htd:aggregate part="NCName"?
1765             location="query"?
1766             condition="bool-expr"?
1767             function="function-call"/>+
```

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

- The optional `part` attribute MUST contain the name of a WSDL part. The `part` attribute MUST be specified when the task interface is defined using a WSDL message with more than one WSDL part.
- The optional `location` attribute MUST contain a query pointing to the location of a leaf element of the tasks' output documents:
  - For each parallel sub task, this is the location of exactly one element of the sub task's output document that is processed by the aggregation function. Each sub tasks' output element is (conditionally) added to a node-set passed as parameter to the aggregation function.
  - For the parent task, this is the element created in the task's output document that is the computed return value of the aggregation function.
- The optional `condition` attribute MUST contain a Boolean expression evaluated on each sub task's output document. If the expression evaluates to `true` then the sub task's output element identified by `location` is added to the node-set passed to the aggregation function.
- The mandatory `function` attribute contains the name of the aggregation function (QName; see a list of supported aggregation functions [below](#) in section 7.2) and optional arguments, in the following form:
 

```
FunctionName '(' ( Argument ( ',' Argument )* )? ')'
```

 Important:
  - The first parameter of each aggregation function is the node-set of sub task's output elements to be aggregated. This parameter is inserted implicitly and MUST NOT be specified within the `function` attribute.
  - Within the `function` attribute, function arguments MUST be specified only for *additional* parameters defined for an aggregation function.

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

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

- Rule (1): If the result value is optional (element defined with `minOccurs="0"` or attribute defined with `use="optional"`) then the corresponding element or attribute in the parent task output MUST be omitted.
- Rule (2): If rule (1) does not apply and a default value is provided (element or attribute defined with `default="{value}"`) then the parent task output element or attribute MUST be explicitly set to this default value.

- 1803 • Rule (3): If rules (1)-(2) do not apply and the result value is a nillable element (element defined  
1804 with `nillable="true"`) then the parent task output element MUST be set to a nil value (`<a  
1805 xsi:nil="true"/>`).
- 1806 • Rule (4): If rules (1)-(3) do not apply, that is, the result is mandatory (element defined with  
1807 `minOccurs="1"` or attribute defined with `use="required"`) but a value cannot be supplied,  
1808 then a standard fault `htd:aggregationFailure` MUST be thrown to indicate a non-  
1809 recoverable error.

### 1810 Example:

1811 Consider the following output document used in a parallel routing pattern:

```
1812 <element name="Award" type="tns:tAward" />
1813 <complexType name="tAward">
1814   <sequence>
1815     <element name="AwardRecommended" type="xsd:string" />
1816     <element name="AwardDetails" type="tns:tAwardDetails" />
1817   </sequence>
1818 </complexType>
1819 <complexType name="tAwardDetails">
1820   <sequence>
1821     <element name="Amount" type="xsd:integer" />
1822     <element name="Appraisal" type="xsd:string" />
1823   </sequence>
1824 </complexType>
```

1825 A possible result aggregation could then look like this. The first aggregation determines the most frequent  
1826 occurrence of an award recommendation. The second aggregation calculates the average award amount  
1827 for sub tasks with an award recommendation of 'yes'. The third aggregation creates a comma-separated  
1828 concatenation of all sub task's appraisals.

```
1829 <htd:parallel ...>
1830   ...
1831   <htd:completionBehavior>
1832     <htd:completion>
1833       <htd:condition> ... </htd:condition>
1834       <htd:result>
1835         <htd:aggregate location="/Award/AwardRecommended"
1836           function="htd:mostFrequentOccurence()" />
1837         <htd:aggregate location="/Award/AwardDetails/Amount"
1838           condition="/Award/AwardRecommended='yes'"
1839           function="htd:avg()" />
1840         <htd:aggregate location="/Award/AwardDetails/Appraisal"
1841           function="htd:concatWithDelimiter(',')" />
1842       </htd:result>
1843     </htd:completion>
1844   </htd:completionBehavior>
1845 </htd:parallel>
```

### 1846 4.8.2.2 Explicit Result Assignment

1847 An `htd:copy` element describes the explicit assignment to an element of the parent task's output  
1848 document.

```
1849 <htd:copy>+
1850   <htd:from expressionLanguage="anyURI"?>
1851     expression
1852   </htd:from>
1853   <htd:to part="NCName"? queryLanguage="anyURI"?>
1854     query
```

```
1855 </htd:to>
1856 </htd:copy>
```

1857 The `htd:copy` element is defined as follows:

- 1858 • The mandatory `htd:from` element MUST contain an expression used to calculate the result  
1859 value. The expression can make use of WS-HumanTask aggregation functions.
- 1860 • The mandatory `htd:to` element MUST contain a query pointing to the location of an element of  
1861 the tasks' output documents. This is the element created in the task's output document.

### 1862 Example 1:

1863 Consider the following output document used in a parallel routing pattern:

```
1864 <element name="Order" type="tns:tOrder" />
1865 <complexType name="tOrder">
1866   <sequence>
1867     <element name="Item" type="tns:tItem" maxOccurs="unbounded"/>
1868     <element name="TotalPrice" type="xsd:integer" />
1869   </sequence>
1870 </complexType>
1871 <complexType name="tItem">
1872   <sequence>
1873     ...
1874   </sequence>
1875 </complexType>
```

1876 A possible result aggregation could then look like this. All sub task order item lists are concatenated to  
1877 one parent task order item list. The total price is calculated using an aggregation function.

```
1878 <htd:parallel>
1879   ...
1880   <htd:completionBehavior>
1881     <htd:completion>
1882       <htd:condition> ... </htd:condition>
1883       <htd:result>
1884         <htd:copy>
1885           <htd:from>
1886             htd:getSubtaskOutputs("orderResponse", "/Order/Item")
1887           </htd:from>
1888           <htd:to>/Order/Item</htd:to>
1889         </htd:copy>
1890         <htd:copy>
1891           <htd:from>
1892             htd:sum(htd:getSubtaskOutputs("orderResponse",
1893               "/Order/TotalPrice"))
1894           </htd:from>
1895           <htd:to>/Order/TotalPrice</htd:to>
1896         </htd:copy>
1897       </htd:result>
1898     </htd:completion>
1899   </htd:completionBehavior>
1900 </htd:parallel>
```

### 1901 Example 2:

1902 Output data from heterogeneous sub tasks is assigned into the parent task's output. The complete  
1903 complex-typed sub task output documents are copied into child elements of the parent task output  
1904 document.

```
1905 <htd:task name="bookTrip">
1906   ... produces itinerary ...
```

```

1907
1908 <htd:composition type="parallel" ...>
1909   <htd:subtask name="bookHotel">
1910     <htd:task>
1911       ... produces hotelReservation ...
1912     </htd:task>
1913   </htd:subtask>
1914   <htd:subtask name="bookFlight">
1915     <htd:task>
1916       ... produces flightReservation ...
1917     </htd:task>
1918   </htd:subtask>
1919 </htd:composition>
1920 ...
1921 <htd:completionBehavior>
1922   <htd:defaultCompletion>
1923     <htd:result>
1924       <htd:copy>
1925         <htd:from>
1926           htd:getSubtaskOutput("bookHotel",
1927                                 "bookHotelResponse",
1928                                 "/hotelReservation")
1929         </htd:from>
1930         <htd:to>/itinerary/hotelReservation</htd:to>
1931       </htd:copy>
1932       <htd:copy>
1933         <htd:from>
1934           htd:getSubtaskOutput("bookFlight",
1935                                 "bookFlightResponse",
1936                                 "/flightReservation")
1937         </htd:from>
1938         <htd:to>/itinerary/flightReservation</htd:to>
1939       </htd:copy>
1940     </htd:result>
1941   </htd:defaultCompletion>
1942 </htd:completionBehavior>
1943 </htd:task>

```

1944  
1945

## 4.9 Elements for Handling Timeouts and Escalations

1946  
1947  
1948  
1949  
1950  
Timeouts and escalations allow the specification of a date or time before which the task or sub task has to reach a specific state. If the timeout occurs a set of actions is performed as the response. The state of the task is not changed. Several deadlines are specified which differ in the point when the timer clock starts and the state which has to be reached with the given duration or by the given date. They are:

- 1951 • Start deadline: Specifies the time until the task has to start, i.e. it has to reach state *InProgress*. It is defined as either the period of time or the point in time until the task has to reach state *InProgress*. Since expressions are allowed, durations and deadlines can be calculated at runtime, which for example enables custom calendar integration. The time starts to be measured from the time at which the task enters the state *Created*. If the task does not reach state *InProgress* by the deadline an escalation action or a set of escalation actions is performed. Once the task is started, the timer becomes obsolete.
- 1952  
1953  
1954  
1955  
1956  
1957
- 1958 • Completion deadline: Specifies the due time of the task. It is defined as either the period of time until the task gets due or the point in time when the task gets due. The time starts to be measured
- 1959

1960 from the time at which the task enters the state *Created*. If the task does not reach one of the final  
1961 states (*Completed, Failed, Error, Exited, Obsolete*) by the deadline an escalation action or a set  
1962 of escalation actions is performed.

1963 The element `<deadlines>` is used to include the definition of all deadlines within the task definition. It is  
1964 optional. If present then the WS-HumanTask Definition MUST specify at least one deadline. Deadlines  
1965 defined in ad-hoc sub tasks created at runtime MUST NOT contradict the deadlines of their parent task.  
1966 The value of the name attribute MUST be unique for all deadline specifications within a task definition.

1967 **Syntax:**

```
1968 <htd:deadlines>  
1969  
1970   <htd:startDeadline name="NCName">*  
1971  
1972     <htd:documentation xml:lang="xsd:language"? >*  
1973       text  
1974     </htd:documentation>  
1975  
1976     ( <htd:for expressionLanguage="anyURI"? >  
1977       duration-expression  
1978     </htd:for>  
1979     | <htd:until expressionLanguage="anyURI"? >  
1980       deadline-expression  
1981     </htd:until>  
1982   )  
1983  
1984   <htd:escalation name="NCName">*  
1985     ...  
1986   </htd:escalation>  
1987 </htd:startDeadline>  
1988  
1989   <htd:completionDeadline name="NCName">*  
1990     ...  
1991   </htd:completionDeadline>  
1992 </htd:deadlines>
```

1995 The language used in expressions is specified using the `expressionLanguage` attribute. This attribute  
1996 is optional. If not specified, the default language as inherited from the closest enclosing element that  
1997 specifies the attribute is used.

1998 For all deadlines if a status is not reached within a certain time then an escalation action, specified using  
1999 element `<escalation>`, can be triggered. The `<escalation>` element is defined in the section below.  
2000 When the task reaches a final state (*Completed, Failed, Error, Exited, Obsolete*) all deadlines are deleted.

2001 Escalations are triggered if

- 2002 1. The associated point in time is reached, or duration has elapsed, and
- 2003 2. The associated condition (if any) evaluates to true

2004 Escalations use notifications to inform people about the status of the task. Optionally, a task might be  
2005 reassigned to some other person or group as part of the escalation. Notifications are explained in more  
2006 detail in section 6 "Notifications". For an escalation, a WS-HumanTask Definition MUST specify exactly  
2007 one escalation action.

2008 When defining escalations, a notification can be either referred to, or defined inline.

- 2009 • A notification defined in the `<humanInteractions>` root element or imported from a different  
2010 namespace can be referenced by specifying its QName in the `reference` attribute of a

2011 <localNotification> element. When referring to a notification, the priority and the people  
 2012 assignments of the original notification definition MAY be overridden using the elements  
 2013 <priority> and <peopleAssignments> contained in the <localNotification> element.

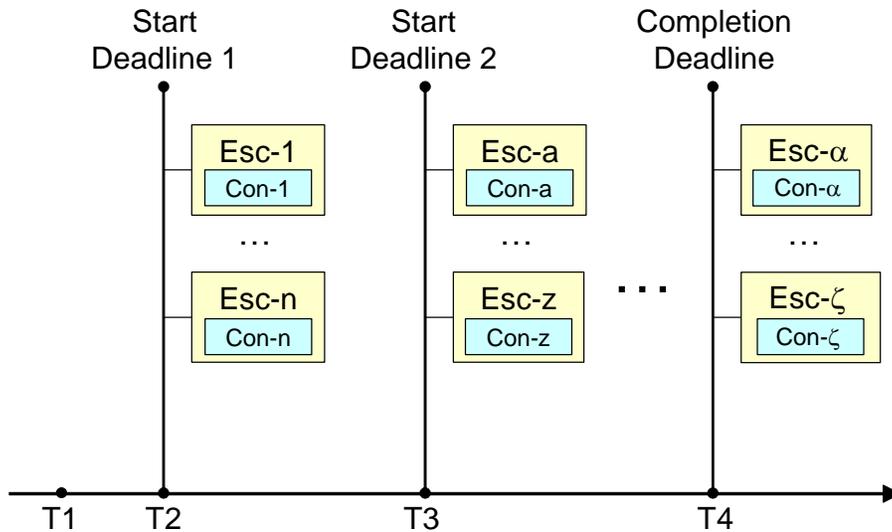
- 2014 • An inlined notification is defined by a <notification> element.

2015 Notifications used in escalations can use the same type of input data as the surrounding task or sub task,  
 2016 or different type of data. If the same type of data is used then the input message of the task or sub task is  
 2017 passed to the notification implicitly. If not, then the <toPart> elements are used to assign appropriate  
 2018 data to the notification, i.e. to explicitly create a multi-part WSDL message from the data. The part  
 2019 attribute refers to a part of the WSDL message. The expressionLanguage attribute specifies the  
 2020 language used in the expression. The attribute is optional. If not specified, the default language as  
 2021 inherited from the closest enclosing element that specifies the attribute is used.

2022 A WS-HumanTask Definition MUST specify a <toPart> element for every part in the WSDL message  
 2023 definition because parts not explicitly represented by <toPart> elements would result in uninitialized parts  
 2024 in the target WSDL message. The order in which parts are specified is not relevant. If multiple <toPart>  
 2025 elements are present, a WS-HumanTask Processor MUST execute them in an “all or nothing” manner. If  
 2026 any of the <toPart>s fails, the escalation action will not be performed and the execution of the task is not  
 2027 affected.

2028 Reassignments are used to replace the potential owners of a task when an escalation is triggered. The  
 2029 <reassignment> element is used to specify reassignment. If present then a WS-HumanTask Definition  
 2030 MUST specify potential owners. A reassignment triggered by a sub task escalation MUST apply to the  
 2031 sub task only. A reassignment MAY comprise of a complex people assignment using Routing Patterns.

2032 In the case where several reassignment escalations are triggered, the first reassignment (lexical order)  
 2033 MUST be considered for execution by the WS-HumanTask Processor. The task is set to state Ready after  
 2034 reassignment. Reassignments and notifications are performed in the lexical order.



2035 A task MAY have multiple start deadlines and completion deadlines associated with it. Each such  
 2036 deadline encompasses escalation actions each of which MAY send notifications to certain people. The  
 2037 corresponding set of people MAY overlap.  
 2038

2039 As an example, the figure depicts a task that has been created at time T1. Its two start deadlines would  
 2040 be missed at time T2 and T3, respectively. The associated escalations whose conditions evaluate to  
 2041 “true” are triggered. Both, the escalations Esc-1 to Esc-n as well as escalations Esc-a to Esc-z can  
 2042 involve an overlapping set of people. The completion deadline would be missed at time T4.

2043 **Syntax:**

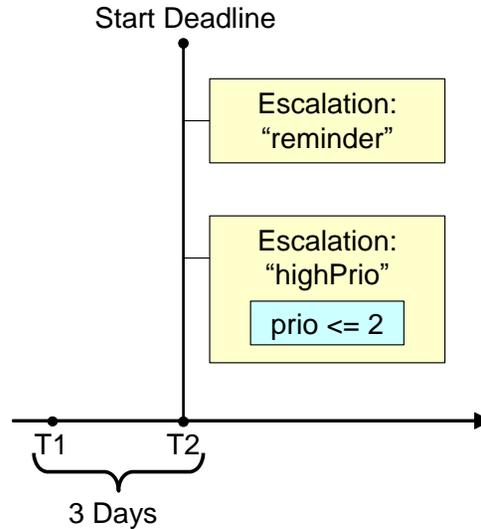
```

2044 <htd:deadlines>
2045
2046   <htd:startDeadline name="NCName">*
2047     ...
2048     <htd:escalation name="NCName">*
2049
2050       <htd:condition expressionLanguage="anyURI"?>?
2051         boolean-expression
2052       </htd:condition>
2053
2054       <htd:toParts>?
2055         <htd:toPart part="NCName"
2056           expressionLanguage="anyURI"?>+
2057           expression
2058         </htd:toPart>
2059       </htd:toParts>
2060
2061       <!-- notification specified by reference -->
2062       <htd:localNotification reference="QName"?
2063         <htd:priority expressionLanguage="anyURI"?>?
2064         integer-expression
2065       </htd:priority>
2066       <htd:peopleAssignments>?
2067         <htd:recipients>
2068           ...
2069         </htd:recipients>
2070       </htd:peopleAssignments>
2071
2072     </htd:localNotification>
2073
2074     <!-- notification specified inline -->
2075     <htd:notification name="NCName">?
2076       ...
2077     </htd:notification>
2078
2079     <htd:reassignment>?
2080
2081       <htd:potentialOwners>
2082         ...
2083       </htd:potentialOwners>
2084
2085     </htd:reassignment>
2086
2087   </htd:escalation>
2088 </htd:startDeadline>
2089
2090   <htd:completionDeadline name="NCName">*
2091     ...
2092   </htd:completionDeadline>
2093
2094 </htd:deadlines>
2095
2096

```

2097 **Example:**

2098 The following example shows the specification of a start deadline with escalations. At runtime, the  
2099 following picture depicts the result of what is specified in the example:



2100 The human task is created at T1. If it has not been started, i.e., no person is working on it until T2, then  
2101 the escalation “reminder” is triggered that notifies the potential owners of the task that work is waiting for  
2102 them. In case the task has high priority then at the same time the regional manager is informed. If the  
2103 task amount is greater than or equal 10000 the task is reassigned to Alan.

2104 In case that task has been started before T2 was reached, then the start deadline is deactivated, no  
2105 escalation occurs.

```
2106 <htd:startDeadline name="sendNotifications">  
2107   <htd:documentation xml:lang="en-US">  
2108     If not started within 3 days, - escalation notifications are sent  
2109     if the claimed amount is less than 10000 - to the task's potential  
2110     owners to remind them or their todo - to the regional manager, if  
2111     this approval is of high priority (0,1, or 2) - the task is  
2112     reassigned to Alan if the claimed amount is greater than or equal  
2113     10000  
2114   </htd:documentation>  
2115   <htd:for>P3D</htd:for>  
2116   <htd:escalation name="reminder">  
2117     <htd:condition>  
2118       <![CDATA[  
2119         htd:getInput("ClaimApprovalRequest")/amount < 10000  
2120         ]]>  
2121     </htd:condition>  
2122     <htd:toParts>  
2123       <htd:toPart name="firstname">  
2124         htd:getInput("ClaimApprovalRequest","ApproveClaim")/firstname  
2125       </htd:toPart>  
2126       <htd:toPart name="lastname">  
2127         htd:getInput("ClaimApprovalRequest","ApproveClaim")/lastname  
2128       </htd:toPart>  
2129     </htd:toParts>  
2130   </htd:escalation>  
2131 </htd:startDeadline>  
2132
```

```

2133 <htd:localNotification reference="tns:ClaimApprovalReminder">
2134
2135   <htd:documentation xml:lang="en-US">
2136     Reuse the predefined notification "ClaimApprovalReminder".
2137     Overwrite the recipients with the task's potential owners.
2138   </htd:documentation>
2139
2140   <htd:peopleAssignments>
2141     <htd:recipients>
2142       <htd:from>htd:getPotentialOwners("ApproveClaim")</htd:from>
2143     </htd:recipients>
2144   </htd:peopleAssignments>
2145
2146 </htd:localNotification>
2147
2148 </htd:escalation>
2149
2150 <htd:escalation name="highPrio">
2151
2152   <htd:condition>
2153     <![CDATA[
2154       (htd:getInput("ClaimApprovalRequest")/amount < 10000
2155       && htd:getInput("ClaimApprovalRequest")/prio <= 2)
2156     ]]>
2157   </htd:condition>
2158
2159   <!-- task input implicitly passed to the notification -->
2160
2161   <htd:notification name="ClaimApprovalOverdue">
2162     <htd:documentation xml:lang="en-US">
2163       An inline defined notification using the approval data as its
2164       input.
2165     </htd:documentation>
2166
2167     <htd:interface portType="tns:ClaimsHandlingPT"
2168       operation="escalate" />
2169
2170     <htd:peopleAssignments>
2171       <htd:recipients>
2172         <htd:from logicalPeopleGroup="regionalManager">
2173           <htd:argument name="region">
2174             htd:getInput("ClaimApprovalRequest")/region
2175           </htd:argument>
2176         </htd:from>
2177       </htd:recipients>
2178     </htd:peopleAssignments>
2179
2180     <htd:presentationElements>
2181       <htd:name xml:lang="en-US">Claim approval overdue</htd:name>
2182       <htd:name xml:lang="de-DE">
2183         Überfällige Schadensforderungsgenehmigung
2184       </htd:name>
2185     </htd:presentationElements>
2186
2187   </htd:notification>
2188
2189 </htd:escalation>
2190

```

```

2191 <htd:escalation name="highAmountReassign">
2192
2193   <htd:condition>
2194     <![CDATA[
2195       htd:getInput("ClaimApprovalRequest")/amount >= 10000
2196     ]]>
2197   </htd:condition>
2198
2199   <htd:reassignment>
2200     <htd:documentation>
2201       Reassign task to Alan if amount is greater than or equal
2202       10000.
2203     </htd:documentation>
2204
2205     <htd:potentialOwners>
2206       <htd:from>
2207         <htd:literal>
2208           <htt:organizationalEntity>
2209             <htt:user>Alan</htt:user>
2210           </htt:organizationalEntity>
2211         </htd:literal>
2212       </htd:from>
2213     </htd:potentialOwners>
2214
2215   </htd:reassignment>
2216
2217 </htd:escalation>
2218
2219 </htd:startDeadline>

```

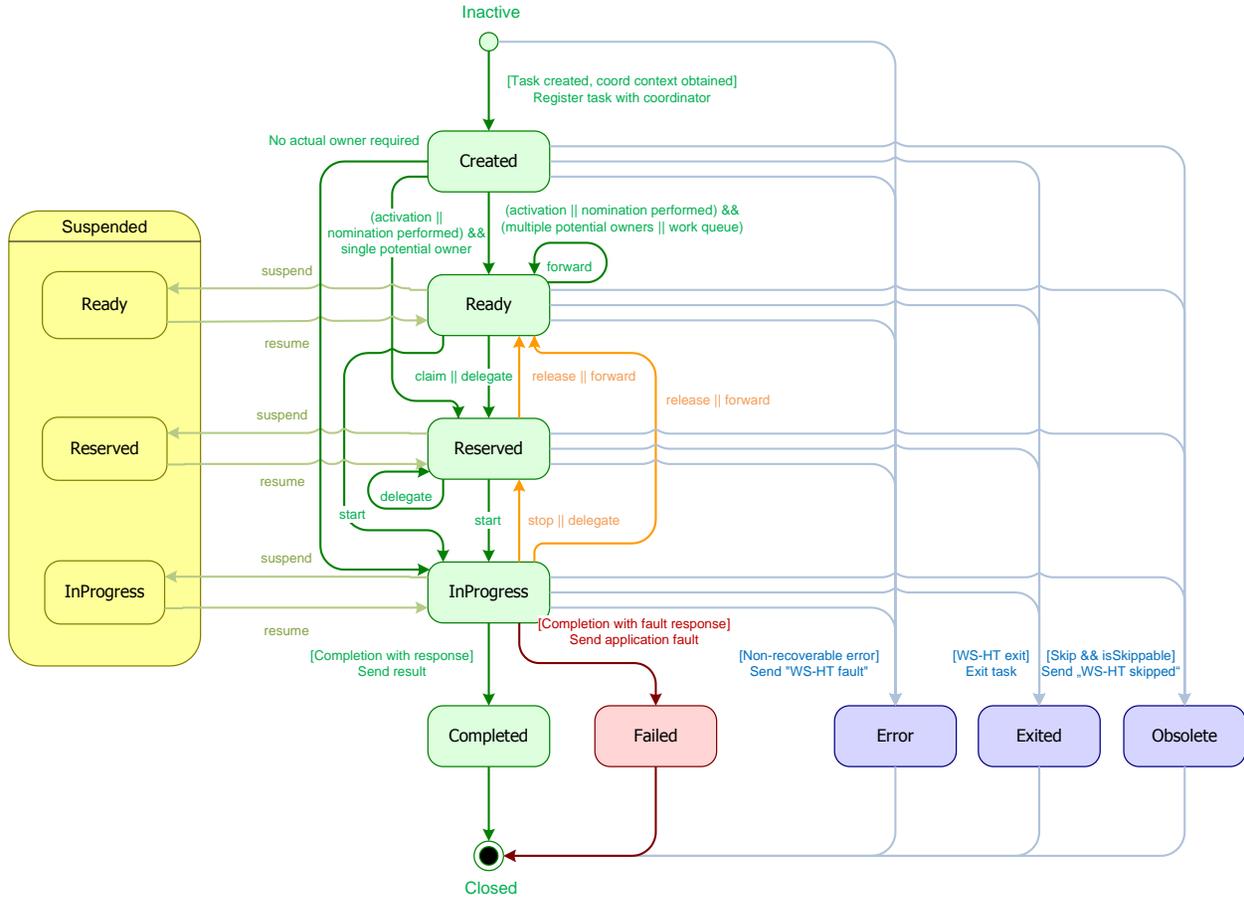
2220 All timeouts and escalations apply to sub tasks also. If htd:escalation is triggered for a sub task, then any  
2221 htd:reassignment **MUST** be applied only to that.

2222

2223

## 2224 4.10 Human Task Behavior and State Transitions

2225 Human tasks can have a number of different states and substates. The state diagram for human tasks  
2226 below shows the different states and the transitions between them.



2227

### 2228 4.10.1 Normal processing of a Human Task

2229 Upon creation, a task goes into its initial state *Created*. Task creation starts with the initialization of its  
2230 properties in the following order:

- 2231 1. Input message
- 2232 2. Priority
- 2233 3. Generic human roles (such as excluded owners, potential owners and business administrators)  
2234 are made available in the lexical order of their definition in the people assignment definition with  
2235 the constraint that excluded owners are taken into account when evaluating the potential owners.
- 2236 4. All other properties are evaluated after these properties in an implementation dependent order.

2237 Task creation succeeds irrespective of whether the people assignment returns a set of values or an  
2238 empty set. People queries that cannot be executed successfully are treated as if they were returning an  
2239 empty set.

2240 If potential owners were not assigned automatically during task creation then they **MUST** be assigned  
2241 explicitly using nomination, which is performed by the task's business administrator. The result of  
2242 evaluating potential owners removes the excluded owners from results. The task remains in the state  
2243 *Created* until it is activated (i.e., an activation timer has been specified) and has potential owners.

2244 When the task has a single potential owner, it transitions into the *Reserved* state, indicating that it is  
2245 assigned to a single actual owner. Otherwise (i.e., when it has multiple potential owners or is assigned to  
2246 a work queue), it transitions into the *Ready* state, indicating that it can be claimed by one of its potential  
2247 owners. Once a potential owner claims the task, it transitions into the *Reserved* state, making that  
2248 potential owner the actual owner.

2249 Once work is started on a task that is in state *Ready* or *Reserved*, it goes into the *InProgress* state,  
2250 indicating that it is being worked on – if the transition is from *Ready*, the user starting the work becomes  
2251 its actual owner.

2252 On successful completion of the work, the task transitions into the *Completed* final state. On unsuccessful  
2253 completion of the work (i.e., with an exception), the task transitions into the *Failed* final state.

2254 The lifecycle of sub tasks is the same as that of the main task.

2255 For human tasks that have subtasks two different cases exist, with different implications:

2256       1. Tasks with subtasks where an actual owner is required

2257       2. Tasks with subtasks where no actual owner is required

2258 The first case has the sub-case where a potential owner has been modeled on the primary task and  
2259 subtasks have been modeled that are activated either manually or automatically. Another sub-case of the  
2260 first case is the one where no potential owner has been modeled and thus nomination has to occur. In all  
2261 cases there is an actual owner eventually and the primary task goes through the state transitions from  
2262 *Created* to *Ready* to *Reserved* to *InProgress*, etc.

2263 In the second case where no actual owner is desired the human task (the primary task) directly transitions  
2264 from state *Created* to *InProgress*. Subtasks are always instantiated automatically.

#### 2265 **4.10.2 Releasing a Human Task**

2266 The current actual owner of a human task can *release* a task to again make it available for all potential  
2267 owners. A task can be released from active states that have an actual owner (*Reserved*, *InProgress*),  
2268 transitioning it into the *Ready* state. Business data associated with the task (intermediate result data, ad-  
2269 hoc attachments and comments) is kept.

2270 A task that is currently *InProgress* can be stopped by the actual owner, transitioning it into state  
2271 *Reserved*. Business data associated with the task as well as its actual owner is kept.

#### 2272 **4.10.3 Delegating or Forwarding a Human Task**

2273 Task's potential owners, actual owner or business administrator can *delegate* a task to another user,  
2274 making that user the actual owner of the task, and also adding her to the list of potential owners in case  
2275 she is not, yet. A task can be delegated when it is in an active state (*Ready*, *Reserved*, *InProgress*), and  
2276 transitions the task into the *Reserved* state. Business data associated with the task is kept.

2277 Similarly, task's potential owners, actual owner or business administrator can forward an active task to  
2278 another person or a set of people, replacing himself by those people in the list of potential owners.  
2279 Potential owners can only forward tasks that are in the *Ready* state. Forwarding is possible if the task has  
2280 a set of individually assigned potential owners, not if its potential owners are assigned using one or many  
2281 groups. If the task is in the *Reserved* or *InProgress* state then the task is implicitly released first, that is,  
2282 the task is transitioned into the *Ready* state. Business data associated with the task is kept. The user  
2283 performing the forward is removed from the set of potential owners of the task, and the forwarder is  
2284 added to the set of potential owners.

#### 2285 **4.10.4 Sub Task Event Propagation**

2286 Task state transitions may be caused by the invocation of API operations (see section 7 "Programming  
2287 Interfaces") or by events (see section 8 "Interoperable Protocol for Advanced Interaction with Human  
2288 Tasks").

2289 If a task has sub tasks then some state transitions are propagated to these sub tasks. Conversely, if a  
 2290 task has a parent task then some state transitions are propagated to that parent task.

2291 The following table defines how task state transitions MUST be propagated to sub tasks and to parent  
 2292 tasks.

| Task Event  | Effect on Sub Tasks<br>(downward propagation)   | Effect on Parent Task<br>(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”) |

2293 All other task state transitions MUST NOT affect sub tasks or a parent task.

2294 **4.11 History of a Human Task**

2295 Task lifecycle state changes and data changes are maintained as a history of task events. Task events  
 2296 contain the following data:

2297 **Task Event**

- 2298 • event id
- 2299 • event time
- 2300 • task id
- 2301 • user (principal) that caused the state change
- 2302 • event type (e.g. claim task).
- 2303 • event data (e.g. data used in setOutput) and fault name (event was setFault)
- 2304 • startOwner - the actual owner before the event.
- 2305 • endOwner - the actual owner after the event.
- 2306 • task status at the end of the event

2307 For example, if the User1 delegated a task to User2, then the user and startOwner would be User1,  
 2308 endOwner would be User2. The event data would be the <htt:organizationalEntity/> element used in the  
 2309 WSHT delegate operation.

2310 The system generated attribute 'event id' MUST be unique on a per task basis.

2311 **4.11.1 Task Event Types and Data**

2312 Some task events (e.g. setOutput) may have data associated with event and others may not (e.g. claim).  
 2313 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          |   |
| <del>Start</del> start                      | maybe        | yes          |   |
| stop  |              | yes          |   |
| release                                     | yes          | yes          |   |
| suspend                                     |              | yes          |   |
| suspendUntil                                |              | yes          | <htt:pointOfTime>2020-12-12T12:12:12Z<br></htt:pointOfTime><br><br>or<br><br><htt:timePeriod>PT1H</htt:timePeriod>  |
| resume                                      |              | yes          |   |
| complete                                    |              | yes          | <htt:taskData><br><ns:someData xmlns:ns="urn:foo"/><br></htt:taskData>  |
| remove                                      |              |              |   |
| fail  |              | yes          | <htt:fail><br><htt:identifier>urn:b4p:1</htt:identifier><br><htt:faultName>fault1</htt:faultName><br><htt:faultData><br><someFaultData xmlns="urn:foo"/><br></htt:faultData><br></htt:fail>                               |
| setPriority                                 |              |              | <htt:priority>500000</htt:priority>   |
| addAttachment                               |              |              | <htt:addAttachment><br><htt:identifier>urn:b4p:1</htt:identifier><br><htt:name>myAttachment</htt:name><br><htt:accessType>MIME</htt:accessType><br><br><htt:contentType>text/plain</htt:contentType><br><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><br><htt:user>user5</htt:user><br><htt:user>user6</htt:user><br></htt:organizationalEntity>   |
| delegate                                    | yes          | maybe        | <htt:organizationalEntity><br><htt:user>user5</htt:user><br></htt:organizationalEntity>   |
| setOutput                                   |              |              | <htt:setOutput><br><htt:identifier>urn:b4p:1</htt:identifier><br><htt:part>outputPart1</htt:part><br><htt:taskData><br><ns:someData xmlns:ns="urn:foo" /><br></htt:taskData><br></htt:setOutput>  |
| deleteOutput                                |              |              |   |
| setFault                                    |              |              | <htt:setFault><br><htt:identifier>urn:b4p:1</htt:identifier><br><htt:faultName>fault1</htt:faultName><br><htt:faultData><someFault<br>xmlns="urn:fault"/></htt:faultData><br></htt:setFault>  |
| deleteFault                                 |              |              |   |
| activate                                    | maybe        | yes          |   |
| nominate                                    | maybe        | maybe        | <htt:organizationalEntity><br><htt:user>user1</htt:user><br><htt:user>user2</htt:user><br></htt:organizationalEntity>   |
| setGenericHumanRole                         |              |              | <htt:setGenericHumanRole><br><htt:identifier>urn:b4p:1</htt:identifier><br><htt:genericHumanRole>businessAdministrators</htt:genericHumanRole><br><htt:organizationalEntity><br><htt:user>user7</htt:user><br><htt:user>user8</htt:user><br></htt:organizationalEntity><br></htt:setGenericHumanRole> |

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

## 2314 4.11.2 Retrieving the History

2315 There is a `getTaskHistory` operation that allows a client to query the system and retrieve a list of task  
2316 events that represent the history of the task. This operation can:

- 2317 • Return a list of task events with optional data
- 2318 • Return a list of task events without optional event data
- 2319 • Return a subset of the events based on a range (for paging)
- 2320 • Return a filtered list of events.

2321 The option to whether or not to include event data is useful since in some cases the event data content  
2322 (e.g. `setOutput`) may be large. In a typical case, an API client should be able to query the system to get a  
2323 "light weight" response of events (e.g. with out event data) and then when necessary, make an additional  
2324 API call to get a specific event details with data. The latter can be accomplished by specifying the event id  
2325 when invoking the `getTaskHistory` operation.

2326 The XML Schema definition of the filter is the following:

```

2327 <xsd:complexType name="tTaskHistoryFilter">
2328   <xsd:choice>
2329     <xsd:element name="eventId" type="xsd:integer" />
2330     <!-- Filter to allow narrow down query by status, principal,
2331          event Type. -->
2332     <xsd:sequence>
2333       <xsd:element name="status" type="tStatus" minOccurs="0"
2334         maxOccurs="unbounded" />
2335       <xsd:element name="eventType" type="tTaskEventType" minOccurs="0"
2336         maxOccurs="unbounded" />
2337       <xsd:element name="principal" type="xsd:string" minOccurs="0" />
2338       <xsd:element name="afterEventTime" type="xsd:dateTime"
2339         minOccurs="0" />
2340       <xsd:element name="beforeEventTime" type="xsd:dateTime"
2341         minOccurs="0" />
2342     </xsd:sequence>
2343   </xsd:choice>
2344 </xsd:complexType>
2345
2346 <xsd:simpleType name="tTaskEventType">
2347   <xsd:restriction base="xsd:string">
2348     <xsd:enumeration value="create" />
2349     <xsd:enumeration value="claim" />
2350     <xsd:enumeration value="start" />
2351     <xsd:enumeration value="stop" />
2352     <xsd:enumeration value="release" />
2353     <xsd:enumeration value="suspend" />

```

```

2354 <xsd:enumeration value="suspendUntil" />
2355 <xsd:enumeration value="resume" />
2356 <xsd:enumeration value="complete" />
2357 <xsd:enumeration value="remove" />
2358 <xsd:enumeration value="fail" />
2359 <xsd:enumeration value="setPriority" />
2360 <xsd:enumeration value="addAttachment" />
2361 <xsd:enumeration value="deleteAttachment" />
2362 <xsd:enumeration value="addComment" />
2363 <xsd:enumeration value="updateComment" />
2364 <xsd:enumeration value="deleteComment" />
2365 <xsd:enumeration value="skip" />
2366 <xsd:enumeration value="forward" />
2367 <xsd:enumeration value="delegate" />
2368 <xsd:enumeration value="setOutput" />
2369 <xsd:enumeration value="deleteOutput" />
2370 <xsd:enumeration value="setFault" />
2371 <xsd:enumeration value="deleteFault" />
2372 <xsd:enumeration value="activate" />
2373 <xsd:enumeration value="nominate" />
2374 <xsd:enumeration value="setGenericHumanRole" />
2375 <xsd:enumeration value="expire" />
2376 <xsd:enumeration value="escalated" />
2377 </xsd:restriction>
2378 </xsd:simpleType>

```

2379 The XML Schema definition of events returned for the history is the following:

```

2380 <xsd:element name="taskEvent">
2381   <xsd:complexType>
2382     <xsd:annotation>
2383       <xsd:documentation>
2384         A detailed event that represents a change in the task's state.
2385       </xsd:documentation>
2386     </xsd:annotation>
2387     <xsd:sequence>
2388       <!-- event id - unique per task -->
2389       <xsd:element name="id" type="xsd:integer" />
2390       <!-- event date time -->
2391       <xsd:element name="eventTime" type="xsd:dateTime" />
2392       <!-- task ID -->
2393       <xsd:element name="identifier" type="xsd:anyURI" />
2394       <xsd:element name="principal" type="xsd:string" minOccurs="0"
2395         nillable="true" />
2396       <!-- Event type. Note - using a restricted type limits
2397         extensibility to add custom event types. -->
2398       <xsd:element name="eventType" type="tTaskEventType" />
2399       <!-- actual owner of the task before the event -->
2400       <xsd:element name="startOwner" type="xsd:string" minOccurs="0"
2401         nillable="true" />
2402       <!-- actual owner of the task after the event -->
2403       <xsd:element name="endOwner" type="xsd:string" minOccurs="0"
2404         nillable="true" />
2405       <!-- WSH task status -->
2406       <xsd:element name="status" type="tStatus" />
2407       <!-- boolean to indicate this event has optional data -->
2408       <xsd:element name="hasData" type="xsd:boolean" minOccurs="0" />
2409       <xsd:element name="eventData" type="xsd:anyType" minOccurs="0"

```

```
2410         nillable="true" />
2411     <xsd:element name="faultName" type="xsd:string" minOccurs="0"
2412         nillable="true" />
2413     <!-- extensibility -->
2414     <xsd:any namespace="##other" processContents="lax" minOccurs="0"
2415         maxOccurs="unbounded" />
2416 </xsd:sequence>
2417 </xsd:complexType>
2418 </xsd:element>
2419
```

---

## 2420 5 Lean Tasks

2421 The <leanTask> element is used to specify human tasks. This section introduces the syntax for the  
2422 element, and individual properties are explained in subsequent sections.

### 2423 5.1 Overall Syntax

2424 The element <leanTask> derives from the type htd:tTask, with the following augmentations:

```
2425 <htd:leanTask>  
2426   <htd:interface>...</htd:interface>  
2427   <htd:messageSchema>...</htd:messageSchema>  
2428   ... All elements from htd:task except <interface> and <composition> ...  
2429   <htd:composition>...</htd:composition>  
2430 </htd:leanTask>
```

### 2431 5.2 Properties

2432 The following attributes and elements are defined for lean tasks and are different from the definition of  
2433 htd:task:

- 2434 • `interface` – Lean tasks are created through the CreateLeanTask operation (section 7.3.4), and  
2435 their input message is derived from the messageSchema element. Therefore, an interface  
2436 element might contradict that information, and to prevent that, interface is banned.
- 2437 • `messageSchema` – Identifies the schema of the inputMessage and outputMessage for the lean  
2438 task, and if the renderings element is not defined, the WS-HumanTask Processor can use this to  
2439 generate a rendering or pass this data directly to a WS-HumanTask Client such that the  
2440 rendering is generated from the messageSchema.
- 2441 • `composition` – Lean tasks cannot have explicitly declared subtasks as defined for composite  
2442 tasks (section 4.6), consequently, this element is banned.

### 2443 5.3 Message Schema

2444 This element references the schema of the data that is used for both the input and output messages of  
2445 the lean task.

```
2446 <messageSchema>  
2447   <messageField name="xsd:NCName" type="xsd:QName"*>  
2448     <messageDisplay xml:lang="xsd:language"?>+  
2449       Language specific display  
2450     </messageDisplay>  
2451     <messageChoice value="xsd:anySimpleType"*>  
2452       <messageDisplay xml:lang="xsd:language"?>+  
2453         Language specific display  
2454       </messageDisplay>  
2455     </messageChoice>  
2456   </messageField>  
2457 </messageSchema>
```

2458 The <messageSchema> element specifies the data that a Lean Task accepts. As it is currently defined, a  
2459 WS-HumanTask Processor could render the following form elements in a way that only requires vendor-  
2460 specific knowledge between the WS-HumanTask Processor and the WS-HumanTask Client and no  
2461 vender-specific knowledge between the WS-HumanTask Processor and the WS-HumanTask Parent:

- 2462 • String
- 2463 • Integer

- 2464 • Float
- 2465 • Date Time
- 2466 • Bool
- 2467 • Enumeration (Choice)

2468 Each of these is accomplished by using an instance of a `<messageField>`. For string, integer, float,  
2469 datetime, and boolean fields, this is accomplished by using the type attribute of the `<messageField>`.  
2470 The supported set of values are: `xsd:string`, `xsd:integer`, `xsd:float`, `xsd:datetime`, and  
2471 `xsd:boolean`, all respectively matching the list above. If a simple rendering language like HTML were  
2472 used, this could be accomplished by using a textbox control that simply had special rules about the format  
2473 of its input.

2474 The enumeration field uses a combination of one `<messageField>` element and possibly many child  
2475 `<messageChoice>` elements. Each child `<messageChoice>` represents one possible option that could  
2476 be selected from the enumeration. If a simple rendering language like HTML were used, this could be  
2477 shown using radio buttons, a dropdown list, or a listbox that only supports single selection.

2478 For all `<messageField>` and `<messageChoice>` elements, it is possible to specify a per-language  
2479 `<messageDisplay>` element. It uses `xml:lang`, a standard XML attribute, to define the language of the  
2480 enclosed information. This attribute uses tags according to RFC 1766 (see [RFC1766]). There could be  
2481 zero or more `<messageDisplay>` elements. A `<messageField>` or `<messageChoice>` MUST NOT  
2482 specify multiple `<messageDisplay>` elements having the same value for the attribute `xml:lang`.

2483 The combination of `<messageSchema>` and `<possibleOutcomes>` can be used to generate a form of  
2484 sufficient functionality for many simple tasks, precluding the need for a renderings element.

#### 2485 **Example:**

```
2486 <messageSchema>
2487   <messageField name="amount" type="xsd:float">
2488     <messageDisplay xml:lang="en-us">Amount</messageDisplay>
2489     <messageDisplay xml:lang="fr-fr">Quantité</messageDisplay>
2490   </messageField>
2491   <messageField name="currencyUnit" type="xsd:string">
2492     <messageDisplay xml:lang="en-us">Currency</messageDisplay>
2493     <messageDisplay xml:lang="fr-fr">Devise</messageDisplay>
2494     <messageChoice value="USD">
2495       <messageDisplay xml:lang="en-us">US Dollars</messageDisplay>
2496       <messageDisplay xml:lang="fr-fr">US Dollars</messageDisplay>
2497     </messageChoice>
2498     <messageChoice value="EURO">
2499       <messageDisplay xml:lang="en-us">Euro Dollars</messageDisplay>
2500       <messageDisplay xml:lang="fr-fr">Euros</messageDisplay>
2501     </messageChoice>
2502   </messageField>
2503 </messageSchema>
```

2504

2505

## 2506 5.4 Example: ToDoTask

2507 The following XML could be used for a simple 'ToDoTask':

```
2508 <htd:task name="ToDoTask">
2509   <htd:messageSchema />
2510   <htd:possibleOutcomes>
2511     <htd:possibleOutcome name="Completed" />
2512     ... language specific translations ...
2513   </htd:possibleOutcomes>
2514   <htd:delegation potentialDelegates="anybody" />
2515   <htd:presentationElements>
2516     <htd:name>ToDo Task</htd:name>
2517     ... language specific translations ...
2518     <htd:subject>Please complete the described work</htd:subject>
2519     ... language specific translations ...
2520     <htd:description contentType="mimeTypeString" />
2521     ... language specific translations ...
2522   </htd:presentationElements>
2523 </htd:task>
```

2524

## 6 Notifications

2525 Notifications are used to notify a person or a group of people of a noteworthy business event, such as  
2526 that a particular order has been approved, or a particular product is about to be shipped. They are also  
2527 used in escalation actions to notify a user that a task is overdue or a task has not been started yet. The  
2528 person or people to whom the notification will be assigned to could be provided, for example, as result of  
2529 a people query to organizational model.

2530 Notifications are simple human interactions that do not block the progress of the caller, that is, the caller  
2531 does not wait for the notification to be completed. Moreover, the caller cannot influence the execution of  
2532 notifications, e.g. notifications are not terminated if the caller terminates. The caller, i.e. an application, a  
2533 business process or an escalation action, initiates a notification passing the required notification data. The  
2534 notification appears on the task list of all notification recipients. After a notification recipient removes it,  
2535 the notification disappears from the recipient's task list.

2536 A notification MAY have multiple recipients and optionally one or many business administrators. The  
2537 generic human roles task initiator, task stakeholders, potential owners, actual owner and excluded  
2538 owners play no role.

2539 Presentation elements and task rendering, as described in sections 4.3 and 4.4 respectively, are used for  
2540 notifications also. In most cases the subject line and description are sufficient information for the  
2541 recipients, especially if the notifications are received in an e-mail client or mobile device. But in some  
2542 cases the notifications can be received in a proprietary client so the notification can support a proprietary  
2543 rendering format to enable this to be utilized to the full, such as for rendering data associated with the  
2544 caller invoking the notification. For example, the description could include a link to the process audit trail  
2545 or a button to navigate to business transactions involved in the underlying process.

2546 Notifications do not have ad-hoc attachments, comments or deadlines.

### 6.1 Overall Syntax

2547 Definition of notifications

```
2548 <htd:notification name="NCName">  
2549   <htd:interface portType="QName" operation="NCName"/>  
2550   <htd:priority expressionLanguage="anyURI"??>  
2551     integer-expression  
2552   </htd:priority>  
2553   <htd:peopleAssignments>  
2554     <htd:recipients>  
2555       ...  
2556     </htd:recipients>  
2557     <htd:businessAdministrators?>  
2558       ...  
2559     </htd:businessAdministrators>  
2560   </htd:peopleAssignments>  
2561   <htd:presentationElements>  
2562     ...  
2563   </htd:presentationElements>  
2564   <htd:renderings?>
```

```
2574 ...
2575 </htd:renderings>
2576
2577 </htd:notification>
```

## 2578 6.2 Properties

2579 The following attributes and elements are defined for notifications:

- 2580 • `name`: This attribute is used to specify the name of the notification. The name combined with the  
2581 target namespace MUST uniquely identify a notification in the notification definition. The attribute  
2582 is mandatory. It is not used for notification rendering.
- 2583 • `interface`: This element is used to specify the operation used to invoke the notification. The  
2584 operation is specified using WSDL, that is a WSDL port type and WSDL operation are defined.  
2585 The element and its `portType` and `operation` attributes are mandatory. In the `operation`  
2586 attribute, a WS-HumanTask Definition MUST reference a one-way WSDL operation.
- 2587 • `priority`: This element is used to specify the priority of the notification. It is an optional  
2588 element which value is an integer expression. If present then the WS-HumanTask Definition  
2589 MUST specify a value between 0 and 10, where 0 is the highest priority and 10 is the lowest. If  
2590 not present, the priority of the notification is considered as 5. The result of the expression  
2591 evaluation is of type `htt:tPriority`. The `expressionLanguage` attribute specifies the  
2592 language used in the expression. The attribute is optional. If not specified, the default language  
2593 as inherited from the closest enclosing element that specifies the attribute is used.
- 2594 • `peopleAssignments`: This element is used to specify people assigned to the notification. The  
2595 element is mandatory. A WS-HumanTask Definition MUST include a people assignment for  
2596 recipients and MAY include a people assignment for business administrators.
- 2597 • `presentationElements`: The element is used to specify different information used to display  
2598 the notification, such as name, subject and description, in a task list. The element is mandatory.  
2599 See section 4.3 for more information on presentation elements.
- 2600 • `rendering`: The element is used to specify rendering method. It is optional. If not present,  
2601 notification rendering is implementation dependent. See section 4.4 for more information on  
2602 rendering.

## 2603 6.3 Notification Behavior and State Transitions

2604 Same as human tasks, notifications are in pseudo-state *Inactive* before they are activated. Once they are  
2605 activated they move to the *Ready* state. This state is observable, that is, when querying for notifications  
2606 then all notifications in state *Ready* are returned. When a notification is removed then it moves into the  
2607 final pseudo-state *Removed*.

---

## 2608 7 Programming Interfaces

### 2609 7.1 Operations for Client Applications

2610 A number of applications are involved in the life cycle of a task. These comprise:

- 2611 • The task list client, i.e. a client capable of displaying information about the task under  
2612 consideration
- 2613 • The requesting application, i.e. any partner that has initiated the task
- 2614 • The supporting application, i.e. an application launched by the task list client to support  
2615 processing of the task.

2616 The task infrastructure provides access to a given task. It is important to understand that what is meant  
2617 by *task list client* is the software that presents a UI to one authenticated user, irrespective of whether this  
2618 UI is rendered by software running on server hardware (such as in a portals environment) or client  
2619 software (such as a client program running on a users workstation or PC).

2620 A given task exposes a set of operations to this end. A WS-HumanTask Processor MUST provide the  
2621 operations listed below and an application (such as a task list client) can use these operations to  
2622 manipulate the task. All operations MUST be executed in a synchronous fashion and MUST return a fault  
2623 if certain preconditions do not hold. For operations that are not expected to return a response they MAY  
2624 return a void message. The above applies to notifications also.

2625 An operation takes a well-defined set of parameters as its input. Passing an illegal parameter or an illegal  
2626 number of parameters MUST result in the `hta:illegalArgumentFault` being returned. Invoking an  
2627 operation that is not allowed in the current state of the task MUST result in an  
2628 `hta:illegalStateFault`.

2629 By default, the identity of the person on behalf of which the operation is invoked is passed to the task.  
2630 When the person is not authorized to perform the operation the `hta:illegalAccessFault` and  
2631 `hta:recipientNotAllowed` MUST be returned in the case of tasks and notifications respectively.

2632 Invoking an operation that does not apply to the task type (e.g., invoking claim on a notification) MUST  
2633 result in an `hta:illegalOperationFault`.

2634 The language of the person on behalf of which the operation is invoked is assumed to be available to  
2635 operations requiring that information, e.g., when accessing presentation elements.

2636 For an overview of which operations are allowed in what state, refer to section 4.10 "Human Task  
2637 Behavior and State Transitions". For a formal definition of the allowed operations, see Appendix D "WS-  
2638 HumanTask Client API Port Type".

2639 For information which generic human roles are authorized to perform which operations, refer to section  
2640 7.1.4 "Operation Authorizations".

2641 This specification does not stipulate the authentication, language passing, addressing, and binding  
2642 scheme employed when calling an operation. This can be achieved using different mechanisms (e.g. WS-  
2643 Security, WS-Addressing).

#### 2644 7.1.1 Participant Operations

2645 Operations are executed by end users, i.e. actual or potential owners. The identity of the user is implicitly  
2646 passed when invoking any of the operations listed in the table below.

2647 If the task is in a predefined state listed as valid pre-state before the operation is invoked then, upon  
2648 successful completion, the task MUST be in the post state defined for the operation. If the task is in a  
2649 predefined state that is not listed as valid pre-state before the operation is invoked then the operation  
2650 MUST be rejected and MUST NOT cause a task state transition.

2651 All of the operations below apply to tasks and sub tasks only unless specifically noted below.

2652 The column “**Supports Batch Processing**” below indicates if an operation can be used to process  
 2653 multiple human tasks at the same time. One or more operations on individual tasks may fail without  
 2654 causing the overall batch operation to fail.  
 2655

| 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"> <li>task identifier</li> <li>attachment name</li> <li>access type</li> <li>content type</li> <li>attachment</li> </ul> Out <ul style="list-style-type: none"> <li>attachment identifier</li> </ul> | 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"> <li>task identifier</li> <li>plain text</li> </ul> Out <ul style="list-style-type: none"> <li>comment identifier</li> </ul>  | 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"> <li>task identifier</li> </ul> Out <ul style="list-style-type: none"> <li>void</li> </ul>  | 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"> <li>task identifier</li> <li>output data of task (optional)</li> </ul> Out <ul style="list-style-type: none"> <li>void</li> </ul>  | 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 <b>MUST</b> be added to the set of potential owners.<br>For details on delegating human tasks refer to section 4.10.3. | In <ul style="list-style-type: none"> <li>task identifier</li> <li>organizational entity<br/>(<code>http:taskOrganizationalEntity</code>)</li> </ul> Out <ul style="list-style-type: none"> <li>void</li> </ul> | Yes                       | Ready<br>Reserved<br>InProgress | Reserved              |
| deleteAttachment | Delete the attachment with the specified identifier from the task.<br>Attachments provided by the enclosing context <b>MUST NOT</b> be affected by this operation.   | In <ul style="list-style-type: none"> <li>task identifier</li> <li>attachment identifier</li> </ul> Out <ul style="list-style-type: none"> <li>void</li> </ul>  | No                        | (any state)                     | (no state transition) |
| deleteComment    | Deletes the identified comment.  | In <ul style="list-style-type: none"> <li>task identifier</li> <li>comment identifier</li> </ul> Out <ul style="list-style-type: none"> <li>void</li> </ul>   | No                        | (any state)                     | (no state transition) |
| deleteFault      | Deletes the fault name and fault data of the task.   | In <ul style="list-style-type: none"> <li>task identifier</li> </ul> Out <ul style="list-style-type: none"> <li>void</li> </ul>   | No                        | InProgress                      | (no state transition) |
| deleteOutput     | Deletes the output data of the task.   | In <ul style="list-style-type: none"> <li>task identifier</li> </ul> Out <ul style="list-style-type: none"> <li>void</li> </ul>   | 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> <b>MUST</b> be returned if the task interface defines no faults.</p> <p>The fault <code>hta:illegalStateFault</code> <b>MUST</b> 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"> <li>task identifier</li> <li>fault (optional) – contains the fault name and fault data</li> </ul> <p>Out</p> <ul style="list-style-type: none"> <li>void</li> </ul>      | Yes                       | InProgress                      | Failed                |
| forward        | <p>Forward the task to another organization entity. The WS-HumanTask Client <b>MUST</b> specify the receiving organizational entity. Potential owners <b>MAY</b> 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"> <li>task identifier</li> <li>organizational entity (<code>htt:tOrganizationalEntity</code>)</li> </ul> <p>Out</p> <ul style="list-style-type: none"> <li>void</li> </ul> | Yes                       | Ready<br>Reserved<br>InProgress | Ready                 |
| getAttachment  | Get the task attachment with the given identifier.   | <p>In</p> <ul style="list-style-type: none"> <li>task identifier</li> <li>attachment identifier</li> </ul> <p>Out</p> <ul style="list-style-type: none"> <li><code>htt:attachment</code></li> </ul>                   | 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"> <li>task identifier</li> </ul> Out <ul style="list-style-type: none"> <li>list of attachment data (list of <code>htt:attachmentInfo</code>)</li> </ul> | No                        | (any state) | (no state transition) |
| getComments        | Get all comments of a task   | In <ul style="list-style-type: none"> <li>task identifier</li> </ul> Out <ul style="list-style-type: none"> <li>list of comments (list of <code>htt:comment</code>)</li> </ul>               | No                        | (any state) | (no state transition) |
| getFault           | Get the fault data of the task.  | In <ul style="list-style-type: none"> <li>task identifier</li> </ul> Out <ul style="list-style-type: none"> <li>fault – contains the fault name and fault data</li> </ul>                    | 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"> <li>task identifier</li> <li>part name (optional for single part messages)</li> </ul> Out <ul style="list-style-type: none"> <li>any type</li> </ul>   | No                        | (any state) | (no state transition) |
| getOutcome         | Get the outcome of the task  | In <ul style="list-style-type: none"> <li>task identifier</li> </ul> Out <ul style="list-style-type: none"> <li>string</li> </ul>  | 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"> <li>task identifier</li> <li>part name (optional for single part messages)</li> </ul> Out <ul style="list-style-type: none"> <li>any type</li> </ul> | No                        | (any state) | (no state transition) |
| getParentTask           | Returns the superior composite task of a sub task   | In <ul style="list-style-type: none"> <li>task identifier</li> </ul> Out <ul style="list-style-type: none"> <li>htt:tTask</li> </ul>   | 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"> <li>task identifier</li> </ul> Out <ul style="list-style-type: none"> <li>task identifier</li> </ul>   | No                        | (any state) | (no state transition) |
| getRendering            | Applies to both tasks and notifications.<br>Returns the rendering specified by the type parameter.              | In <ul style="list-style-type: none"> <li>task identifier</li> <li>rendering type</li> </ul> Out <ul style="list-style-type: none"> <li>any type</li> </ul>                                | No                        | (any state) | (no state transition) |
| getRenderingTypes       | Applies to both tasks and notifications.<br>Returns the rendering types available for the task or notification. | In <ul style="list-style-type: none"> <li>task identifier</li> </ul> Out <ul style="list-style-type: none"> <li>list of QNames</li> </ul>  | 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"> <li>task identifier</li> </ul> Out <ul style="list-style-type: none"> <li>list of task identifiers</li> </ul>  | No                        | (any state) | (no state transition) |
| getSubtasks             | Returns all sub tasks of a task (created instances + not yet created sub task definitions)                      | In <ul style="list-style-type: none"> <li>task identifier</li> </ul> Out <ul style="list-style-type: none"> <li>list of tasks (list of htt:tTask)</li> </ul>                               | 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"> <li>task identifier</li> <li>content type – optional, default is text/plain</li> </ul> Out <ul style="list-style-type: none"> <li>string</li> </ul>  | 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"> <li>task identifier</li> </ul> Out <ul style="list-style-type: none"> <li>task (<code>htt:tTaskDetails</code>)</li> </ul>  | No                        | (any state) | (no state transition) |
| getTaskHistory     | Get a list of events representing the history of the task.<br><i>Filter</i> allows narrowing the results by status, principal, event Type.<br><i>startIndex</i> and <i>maxTasks</i> are integers that allow paging of the results.<br><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"> <li>task identifier</li> <li>filter (<code>htt:tTaskHistoryFilter</code>)</li> <li>startIndex</li> <li>maxTasks</li> <li>includeData</li> </ul> Out <ul style="list-style-type: none"> <li>list of <code>htt:taskEvent</code></li> </ul> | 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"> <li>• task identifier</li> <li>• properties</li> <li>• rendering preferences</li> </ul> <p>Out</p> <ul style="list-style-type: none"> <li>• task (http:taskInstanceData)</li> </ul> | 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"> <li>task identifier</li> </ul> Out <ul style="list-style-type: none"> <li>List of available operation.</li> </ul>          | 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"> <li>task identifier</li> </ul> Out <ul style="list-style-type: none"> <li>boolean</li> </ul>                               | No                        | (any state)             | (no state transition) |
| instantiateSubTask | Creates an instantiateable subtask for the task from the definition of the task.<br><br>The fault <code>hta:illegalArgumentFault</code> MUST be returned if the task does not have an instantiateable subtask of the given name.<br><br>Returns the identifier for the created subtask. | In <ul style="list-style-type: none"> <li>task identifier</li> <li>subtask name</li> </ul> Out <ul style="list-style-type: none"> <li>task identifier</li> </ul> | No                        | Reserved<br>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"> <li>task identifier</li> </ul> Out <ul style="list-style-type: none"> <li>boolean</li> </ul>                               | 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"> <li>task identifier</li> </ul> Out <ul style="list-style-type: none"> <li>void</li> </ul>                                  | Yes                       | InProgress<br>Reserved  | Ready                 |

| Operation Name | Description   | Parameters  | Supports Batch Processing | Pre-State   | Post-State   |
|----------------|---|---|---------------------------|---|--|
| remove         | Applies to notifications only.<br>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"> <li>task identifier</li> </ul> Out <ul style="list-style-type: none"> <li>void</li> </ul>   | Yes                       | Ready (Notification state)                                    | Removed (Notification state)   |
| resume         | Resume a suspended task.  | In <ul style="list-style-type: none"> <li>task identifier</li> </ul> Out <ul style="list-style-type: none"> <li>void</li> </ul>   | Yes                       | Suspended/Ready<br>Suspended/Reserved<br>Suspended/InProgress | Ready (from Suspended/Ready)<br>Reserved (from Suspended/Reserved)<br>InProgress (from Suspended/InProgress) |
| setFault       | Set the fault data of the task.<br>The fault <code>hta:illegalOperationFault</code> MUST be returned if the task interface defines no faults.   | In <ul style="list-style-type: none"> <li>task identifier</li> <li>fault – contains the fault name and fault data</li> </ul> Out <ul style="list-style-type: none"> <li>void</li> </ul>                             | 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"> <li>task identifier</li> <li>part name (optional for single part messages)</li> <li>output data of task</li> </ul> Out <ul style="list-style-type: none"> <li>void</li> </ul> | 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"> <li>task identifier</li> <li>priority (http:priority)</li> </ul> Out <ul style="list-style-type: none"> <li>void</li> </ul>                   | 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"> <li>task identifier</li> <li>deadline name</li> <li>deadline expression</li> </ul> Out <ul style="list-style-type: none"> <li>void</li> </ul> | Yes                       | Created<br>Ready<br>Reserved<br>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"> <li>task identifier</li> <li>deadline name</li> <li>duration expression</li> </ul> Out <ul style="list-style-type: none"> <li>void</li> </ul> | Yes                       | Created<br>Ready<br>Reserved<br>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"> <li>task identifier</li> <li>deadline name</li> <li>deadline expression</li> </ul> Out <ul style="list-style-type: none"> <li>void</li> </ul> | Yes                       | Created<br>Ready<br>Reserved<br>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"> <li>task identifier</li> <li>deadline name</li> <li>duration expression</li> </ul> Out <ul style="list-style-type: none"> <li>void</li> </ul> | Yes                       | Created<br>Ready<br>Reserved<br>In Progress | (no state transition) |

| Operation Name | Description  | Parameters  | Supports Batch Processing | Pre-State                                  | Post-State   |
|----------------|--|---|---------------------------|--|--|
| skip           | Skip the task.<br>If the task is not skipable then the fault <code>hta:illegalOperationFault</code> MUST be returned.  | In <ul style="list-style-type: none"> <li>task identifier</li> </ul> Out <ul style="list-style-type: none"> <li>void</li> </ul>   | Yes                       | Created<br>Ready<br>Reserved<br>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"> <li>task identifier</li> </ul> Out <ul style="list-style-type: none"> <li>void</li> </ul>   | Yes                       | Ready<br>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"> <li>task identifier</li> </ul> Out <ul style="list-style-type: none"> <li>void</li> </ul>   | Yes                       | InProgress                                 | Reserved   |
| suspend        | Suspend the task.  | In <ul style="list-style-type: none"> <li>task identifier</li> </ul> Out <ul style="list-style-type: none"> <li>void</li> </ul>   | Yes                       | Ready<br>Reserved<br>InProgress            | Suspended/Ready (from Ready)<br>Suspended/Reserved (from Reserved)<br>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"> <li>task identifier</li> <li>time period</li> <li>point of time</li> </ul> Out <ul style="list-style-type: none"> <li>void</li> </ul> | Yes                       | Ready<br>Reserved<br>InProgress            | Suspended/Ready (from Ready)<br>Suspended/Reserved (from Reserved)<br>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"> <li>task identifier</li> <li>comment identifier</li> <li>plain text</li> </ul> Out <ul style="list-style-type: none"> <li>void</li> </ul> | No                        | (any state) | (no state transition) |

2656

## 2657 7.1.2 Simple Query Operations

2658 Simple query operations allow retrieving task data. These operations MUST be supported by a WS-  
 2659 HumanTask Processor. The identity of the user is implicitly passed when invoking any of the following  
 2660 operations.

2661 The following operations will return both matching tasks and sub tasks.

2662

| 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> ("&lt;&gt;"), <i>less than</i> ("&lt;"), <i>greater than</i> ("&gt;"), <i>less than or equals</i> ("&lt;="), <i>and greater than or equals</i> ("&gt;="), e.g., "Task.Priority = 1". <i>greater than or equals</i> ("&gt;="), 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".</p> | In <ul style="list-style-type: none"> <li>task type ("ALL"   "TASKS"   "NOTIFICATIONS")</li> <li>generic human role</li> <li>work queue</li> <li>status list</li> <li>where clause</li> <li>order-by clause</li> <li>created-on clause</li> <li>maxTasks</li> <li>taskIndexOffset</li> </ul> Out <ul style="list-style-type: none"> <li>list of tasks (list of <code>htt:tTaskAbstract</code>)</li> </ul> | Any           |

| Operation Name   | Description  | Parameters  | Authorization |
|------------------|--|---|---------------|
|                  | <p><u>A value of type xsd:QName MUST be 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'".</u></p> <p><u>A comparison with a value of type htt:OrganizationalEntity MUST be performed using its user/group child elements. An example is "task.potentialOwner.user IN ( 'Joe', 'Fred' ) OR task.potentialOwner.group = 'approvers'".</u></p> <p>The created-on clause is optional. The <i>where</i> clause is logically ANDed with the created-on clause, which MUST reference the column Task.CreatedTime with operators as described above.</p> <p>The combination of the two clauses enables simple but restricted paging in a task list client.</p> <p>If maxTasks is specified, then the number of task abstracts returned for this query MUST NOT exceed this limit. The taskIndexOffset can be used to perform multiple identical queries and iterate over result sets where the maxTasks size exceeds the query limit. If maxTasks has not been specified then all tasks fulfilling the query are returned.</p> |   |               |
| getMyTaskDetails | <p>Retrieve the task details. This operation is used to obtain the data required to display a task list, as well as the details for the individual tasks.</p> <p>If no task type has been specified then the default value "ALL" MUST be used.</p> <p>If no generic human role has been specified then the default value "actualOwner" MUST be used.</p> <p>If no work queue has been specified then only personal tasks MUST be returned. If the work queue is specified then only tasks of that work queue MUST be returned.</p> <p>If no status list has been specified then tasks in all valid states are</p>  | <p>In</p> <ul style="list-style-type: none"> <li>• task type ("ALL"   "TASKS"   "NOTIFICATIONS")</li> <li>• generic human role</li> <li>• work queue</li> <li>• status list</li> <li>• where clause</li> <li>• created-on clause</li> <li>• maxTasks</li> </ul> <p>Out</p> <ul style="list-style-type: none"> <li>• list of tasks (list of htt:tTaskDetails)</li> </ul> | Any           |

| Operation Name | Description   | Parameters | Authorization |
|----------------|---|------------|---------------|
|                | <p>returned.</p> <p>The where clause is optional. <u>If specified, it MUST reference exactly one column using the following operators: equals ("="), not equals ("&lt;&gt;"), less than ("&lt;"), greater than ("&gt;"), less than or equals ("&lt;="), and greater than or equals ("&gt;="), e.g., "Task.Priority = 1". If specified, it MUST follow the same rules described for the <a href="#">getMyTaskAbstracts operation</a>.</u></p> <p>The created-on clause is optional. The <i>where</i> clause is logically ANDed with the created-on clause, which MUST reference the column Task.CreatedTime with operators as described above.</p> <p>The combination of the two clauses enables simple but restricted paging in the task list client.</p> <p>If maxTasks is specified, then the number of task details returned for this query MUST NOT exceed this limit. If maxTasks has not been specified then all tasks fulfilling the query are returned.</p> |            |               |

2663

2664 The return types `tTaskAbstract` and `tTaskDetails` are defined in section 3.8.4 "Data Types for Task  
2665 Instance Data".

### 2666 Simple Task View

2667 The table below lists the task attributes available to the simple query operations. This view is used when  
2668 defining the where clause of any of the above query operations.

2669

| Column Name    | Type  |
|----------------|---|
| ID             | xsd:stringanyURI  |
| TaskType       | Enumeration   |
| Name           | xsd:QName   |
| Status         | Enumeration (for values see 4.10 "Human Task Behavior and State Transitions") |
| Priority       | htt:tPriority   |
| CreatedTime    | xsd:dateTime  |
| ActivationTime | xsd:dateTime  |

| Column Name           | Type             |
|-----------------------|------------------|
| ExpirationTime        | xsd:dateTime     |
| HasPotentialOwners    | xsd:boolean      |
| StartByTimeExists     | xsd:boolean      |
| CompleteByTimeExists  | xsd:boolean      |
| RenderingMethodExists | xsd:boolean      |
| Escalated             | xsd:boolean      |
| ParentTaskId          | xsd:stringanyURI |
| HasSubTasks           | xsd:boolean      |
| SearchBy              | xsd:string       |
| Outcome               | xsd:string       |

2670

### 2671 7.1.3 Advanced Query Operation

2672 The advanced query operation is used by the task list client to perform queries not covered by the simple  
 2673 query operations defined in 7.1.2. A WS-HumanTask Processor MAY support this operation. An  
 2674 implementation MAY restrict the results according to authorization of the invoking user.

2675

2676 The following operations will return both matching tasks and sub tasks.

2677

| 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><a href="#">For data of type xsd:QName Or htt:tOrganizationalEntity in a where clause, see the description of the getMyTaskAbstracts operation in section 7.1.2.</a></p> | <p>In</p> <ul style="list-style-type: none"> <li>• select clause</li> <li>• where clause</li> <li>• order-by clause</li> <li>• maxTasks</li> <li>• taskIndexOffset</li> </ul> <p>Out</p> <ul style="list-style-type: none"> <li>• task query result set (htt:tTaskQueryResults et)</li> </ul> |

2678

2679 **ResultSet Data Type**

2680 This is the result set element that is returned by the query operation.

```
2681 <xsd:element name="taskQueryResultSet" type="tTaskQueryResultSet" />
2682 <xsd:complexType name="tTaskQueryResultSet">
2683   <xsd:sequence>
2684     <xsd:element name="row" type="tTaskQueryResultRow"
2685       minOccurs="0" maxOccurs="unbounded" />
2686   </xsd:sequence>
2687 </xsd:complexType>
2688
```

2689 The following is the type of the row element contained in the result set. The value in the row are returned  
2690 in the same order as specified in the select clause of the query.

```
2691 <xsd:complexType name="tTaskQueryResultRow">
2692   <xsd:choice minOccurs="0" maxOccurs="unbounded">
2693     <xsd:element name="id" type="xsd:anyURI"/>
2694     <xsd:element name="taskType" type="xsd:string"/>
2695     <xsd:element name="name" type="xsd:QName"/>
2696     <xsd:element name="status" type="tStatus"/>
2697     <xsd:element name="priority" type="htt:tPriority"/>
2698     <xsd:element name="taskInitiator"
2699       type="htt:tUser"/>
2700     <xsd:element name="taskStakeholders"
2701       type="htt:tOrganizationalEntity"/>
2702     <xsd:element name="potentialOwners"
2703       type="htt:tOrganizationalEntity"/>
2704     <xsd:element name="businessAdministrators"
2705       type="htt:tOrganizationalEntity"/>
2706     <xsd:element name="actualOwner" type="htt:tUser"/>
2707     <xsd:element name="notificationRecipients"
2708       type="htt:tOrganizationalEntity"/>
2709     <xsd:element name="createdTime" type="xsd:dateTime"/>
2710     <xsd:element name="createdBy" type="xsd:stringtUser"/>
2711     <xsd:element name="lastModifiedTime" type="xsd:dateTime"/>
2712     <xsd:element name="lastModifiedBy" type="xsd:stringtUser"/>
2713     <xsd:element name="activationTime" type="xsd:dateTime"/>
2714     <xsd:element name="expirationTime" type="xsd:dateTime"/>
2715     <xsd:element name="isSkipable" type="xsd:boolean"/>
2716     <xsd:element name="hasPotentialOwners" type="xsd:boolean"/>
2717     <xsd:element name="startByTime" type="xsd:dateTime"/>
2718     <xsd:element name="completeByTime" type="xsd:dateTime"/>
2719     <xsd:element name="presentationName" type="tPresentationName"/>
2720     <xsd:element name="presentationSubject"
2721       type="tPresentationSubject"/>
2722     <xsd:element name="renderingMethodName" type="xsd:QName"/>
2723     <xsd:element name="hasOutput" type="xsd:boolean"/>
2724     <xsd:element name="hasFault" type="xsd:boolean"/>
2725     <xsd:element name="hasAttachments" type="xsd:boolean"/>
2726     <xsd:element name="hasComments" type="xsd:boolean"/>
2727     <xsd:element name="escalated" type="xsd:boolean"/>
2728     <xsd:element name="parentTaskId" type="xsd:anyURI"/>
2729     <xsd:element name="hasSubTasks" type="xsd:boolean"/>
2730     <xsd:element name="searchBy" type="xsd:string"/>
2731     <xsd:element name="outcome" type="xsd:string"/>
2732     <xsd:element name="taskOperations" type="tTaskOperations"/>
2733     <xsd:any namespace="##other" processContents="lax"/>
2734   </xsd:choice>
2735 </xsd:complexType>
```

2736 |

2737 |

2738 **Complete Task View**

2739 The table below is the set of columns used when defining select clause, where clause, and order-by  
2740 clause of query operations. Conceptually, this set of columns defines a universal relation. As a result the  
2741 query can be formulated without specifying a from clause. A WS-HumanTask Processor MAY extend this  
2742 view by adding columns.

2743 |

| Column Name        | Type   | Constraints  |
|--------------------|--|--|
| ID                 | xsd: <del>string</del> <u>anyURI</u>                     |  |
| TaskType           | Enumeration  | Identifies the task type. The following values are allowed: <ul style="list-style-type: none"> <li>• “TASK” for a human task</li> <li>• “NOTIFICATION” for notifications</li> </ul> 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) | <del>xsd</del> htt:tUser or<br>htt:tOrganizationalEntity |  |
| CreatedTime        | xsd:dateTime   | The time in UTC when the task has been created.  |
| CreatedBy          | <del>xsd:string</del> htt:tUser                          |  |
| LastModifiedTime   | xsd:dateTime   | The time in UTC when the task has been last modified.  |
| LastModifiedBy     | <del>xsd:string</del> 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.   |
| CompleteByTime     | xsd:dateTime   | The time in UTC when the task needs to be completed. This time corresponds to the respective end deadline.   |

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

2745

## 2746 7.1.4 Administrative Operations

2747 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<br>task identifier<br>Out<br>void   | Yes                       | Created   | Ready             |
| nominate       | Nominate an organization entity to process the task. If it is nominated to one person then the new state of the task is <i>Reserved</i> . If it is nominated to several people then the new state of the task is | In<br>task identifier<br>organizational entity<br>(htt:tOrganizationalEntity)<br>Out<br>void | Yes                       | Created   | Ready<br>Reserved |

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

2748

2749

2750 **7.1.5 Operation Authorizations**

2751 The table below summarizes the required authorizations in terms of generic human roles to execute  
 2752 participant, query and administrative operations. Thus, it is a precise definition of the generic human roles  
 2753 as well. The sign plus ('+') means that the operation MUST be available for the generic human role. The  
 2754 sign minus ('-') means that the operation MUST NOT be available for the generic human role. 'n/a'  
 2755 indicates that the operation is not applicable and thus MUST NOT be available for the generic human  
 2756 role. 'MAY' defines that vendor MAY chose to support the operation for the generic human role.

2757 If a person has multiple generic human roles on a human task or notification and she is allowed to  
 2758 perform an operation in any of the roles then the invocation of the operation will not fail, otherwise  
 2759 hta:illegalAccessFault and hta:recipientNotAllowed MUST be returned in the case of tasks  
 2760 and notifications respectively. If a person is included in the list of excluded owners of a task then she  
 2761 MUST NOT perform any of the operations.

2762 All batch operations (operations with a name prefix "batch") may be invoked by any caller; no specific  
 2763 authorization is required. Missing authorizations for operations on individual tasks result in a report entry  
 2764 in the batch operation's response message.

2765

| 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                     |

2767

## 2768 7.2 XPath Extension Functions

2769 This section introduces XPath extension functions that are provided to be used within the definition of a  
 2770 human task or notification. A WS-HumanTask Processor MUST support the XPath Functions listed below.  
 2771 When defining properties using these XPath functions, note the initialization order in section 4.10.1.

2772 Definition of these XPath extension functions is provided in the table below. Input parameters that specify  
 2773 task name, message part name or logicalPeopleGroup name MUST be literal strings. This restriction  
 2774 does not apply to other parameters. Because XPath 1.0 functions do not support returning faults, an  
 2775 empty node set is returned in the event of an error.

2776 XPath functions used for notifications in an escalation can access context from the enclosing task by  
 2777 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.<br>If the task name is not present the current task MUST be considered.                  | In <ul style="list-style-type: none"> <li>task name (optional)</li> </ul> Out <ul style="list-style-type: none"> <li>the actual owner (user id as <code>htt:user</code>)</li> </ul>   |
| getBusinessAdministrators     | Returns the business administrators of the task.<br>It MUST evaluate to an empty <code>htt:organizationalEntity</code> in case of an error.<br>If the task name is not present the current task MUST be considered. | In <ul style="list-style-type: none"> <li>task name (optional)</li> </ul> Out <ul style="list-style-type: none"> <li>business administrators (<code>htt:organizationalEntity</code>)</li> </ul>   |
| getCountOfFinishedSubTasks    | Returns the number of finished sub tasks of a task<br>If the task name is not present the current task MUST be considered   | In <ul style="list-style-type: none"> <li>task name (optional)</li> </ul> Out <ul style="list-style-type: none"> <li>Number of the finished task sub-tasks. If the task doesn't have sub tasks then 0 is returned</li> </ul>                              |
| getCountOfSubTasks            | Returns the number of sub tasks of a task<br>If the task name is not present the current task MUST be considered  | In <ul style="list-style-type: none"> <li>task name (optional)</li> </ul> Out <ul style="list-style-type: none"> <li>Number of the task sub-tasks. If the task doesn't have sub tasks then 0 is returned</li> </ul>                                       |
| getCountOfSubTasksInState     | Returns the number of a task subtasks that are in the specified state<br>If the task name is not present the current task MUST be considered  | In <ul style="list-style-type: none"> <li>state</li> <li>task name (optional)</li> </ul> Out <ul style="list-style-type: none"> <li>Number of the task sub tasks in the specified state. If the task doesn't have sub tasks then 0 is returned</li> </ul> |
| getCountOfSubTasksWithOutcome | Returns the number of a task sub tasks that match the given outcome<br>If the task name is not present the current task   | In <ul style="list-style-type: none"> <li>outcome</li> <li>task name (optional)</li> </ul> Out  |

| Operation Name        | Description   | Parameters   |
|-----------------------|---|--|
|                       | MUST be considered  | <ul style="list-style-type: none"> <li>Number of the task sub tasks that match the specified outcome. If the task doesn't have sub tasks then 0 is returned</li> </ul>   |
| 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"> <li>task name (optional)</li> </ul> <p>Out</p> <ul style="list-style-type: none"> <li>excluded owners (<code>htt:organizationalEntity</code>)</li> </ul>  |
| 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"> <li>part name</li> <li>task name (optional)</li> </ul> <p>Out</p> <ul style="list-style-type: none"> <li>input message part</li> </ul>  |
| 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"> <li>name of the logical people group</li> <li>The optional parameters that follow MUST appear in pairs. Each pair is defined as: <ul style="list-style-type: none"> <li>the qualified name of a logical people group parameter</li> <li>the value for the named logical people group parameter; it can be an XPath expression</li> </ul> </li> </ul> <p>Out</p> <ul style="list-style-type: none"> <li>the value of the logical people group (<code>htt:organizationalEntity</code>)</li> </ul> |
| 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"> <li>task name (optional)</li> </ul> <p>Out</p> <ul style="list-style-type: none"> <li>the task outcome (<code>xsd:string</code>)</li> </ul>   |

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

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

2779

2780 Generic set functions:

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

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

2781

2782 In addition to the general-purpose functions listed above, the following aggregation functions MUST be  
2783 supported by a WS-HumanTask Processor. All aggregation functions take a node-set of strings,  
2784 booleans, or numbers as the first input parameter, and produce a result of the same type.

2785 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"> <li>node-set of string nodes</li> </ul>                           |
| 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"> <li>node-set of string nodes</li> <li>delimiter string</li> </ul> |
| 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"> <li>node-set of string nodes</li> </ul>                           |
| 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"> <li>node-set of string nodes</li> </ul>                           |
| 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"> <li>node-set of string nodes</li> <li>percentage</li> </ul>       |

2786

2787

2788

2789

2790

2791 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"> <li>node-set of boolean nodes</li> </ul>                     |
| or             | Returns the disjunction of all boolean nodes - returns false for an empty node-set  | In <ul style="list-style-type: none"> <li>node-set of boolean nodes</li> </ul>                     |
| 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"> <li>node-set of boolean nodes</li> <li>percentage</li> </ul> |

2792

2793 Number-valued aggregation functions:

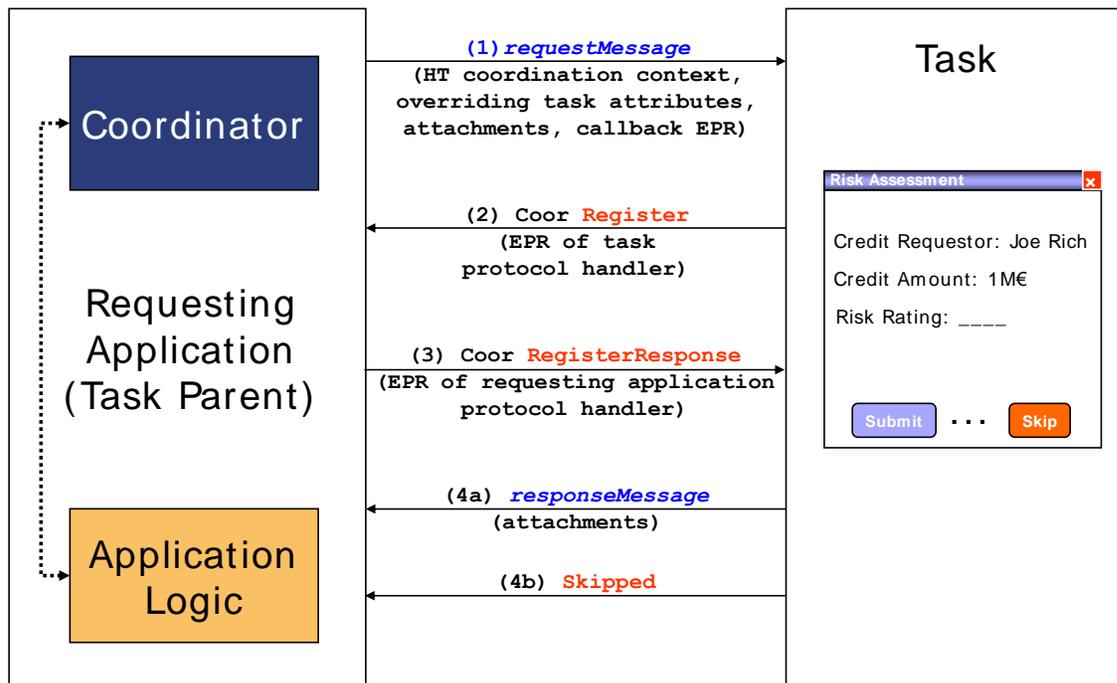
| Operation Name | Description   | Parameters  |
|----------------|---|---|
| avg            | Returns the average value of all number nodes - returns NaN for an empty node-set | In <ul style="list-style-type: none"> <li>node-set of number nodes</li> </ul> |
| max            | Returns the maximum value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

2794

2795  
2796  
2797  
2798  
2799  
2800  
2801  
2802  
2803

## 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.



2804

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

2805 While we do not make any assumptions about the nature of the application in the following scenarios, in  
2806 practice it would be hosted by an infrastructure that actually deals with the WS-HumanTask coordination  
2807 protocol on the application's behalf.

2808 In case of human tasks the following message exchanges are possible.

2809 **Scenario 1:** At some point in time, the application invokes the human task through its service interface. In  
2810 order to signal to the WS-HumanTask Processor that an instance of the human task can be created  
2811 which is actually coordinated by the parent application, this request message contains certain control  
2812 information. This control information consists of a coordination context of the WS-HumanTask  
2813 coordination protocol, and optional human task attributes that are used to override aspects of the human  
2814 task definition.

- 2815 • The coordination context (see [WS-C] for more details on Web services coordination framework  
2816 used here) contains the element `CoordinationType` that MUST specify the WS-HumanTask  
2817 coordination type [http://docs.oasis-open.org/ns/bpel4people/ws-](http://docs.oasis-open.org/ns/bpel4people/ws-humantask/protocol/200803)  
2818 [humantask/protocol/200803](http://docs.oasis-open.org/ns/bpel4people/ws-humantask/protocol/200803). The inclusion of a coordination context within the request

2819 message indicates that the life cycle of the human tasks is managed via corresponding protocol  
2820 messages from outside the WS-HumanTask Processor. The coordination context further contains  
2821 in its `RegistrationService` element an endpoint reference that the WS-HumanTask  
2822 Processor MUST use to register the task as a participant of that coordination type.  
2823 Note: In a typical implementation, the parent application or its environment will create that  
2824 coordination context by issuing an appropriate request against the WS-Coordination (WS-C)  
2825 activation service, followed by registering the parent application as a `TaskParent` participant in  
2826 that protocol.

- 2827 • The optional human task attributes allow overriding aspects of the definition of the human task  
2828 from the calling application. The WS-HumanTask Parent MAY set values of the following  
2829 attributes of the task definition:
  - 2830 ○ Priority of the task
  - 2831 ○ Actual people assignments for each of the generic human roles of the human task
  - 2832 ○ The skipable indicator which determines whether a task can actually be skipped at  
2833 runtime.
  - 2834 ○ The amount of time by which the task activation is deferred.
  - 2835 ○ The expiration time for the human task after which the calling application is no longer  
2836 interested in its result.

2837 After having created this request message, it is sent to the WS-HumanTask Processor (step (1) in Figure  
2838 10). The WS-HumanTask Processor receiving that message MUST extract the coordination context and  
2839 callback information, the human task attributes (if present) and the application payload. Before applying  
2840 this application payload to the new human task, the WS-HumanTask Processor MUST register the human  
2841 task to be created with the registration service passed as part of the coordination context (step (2) in  
2842 Figure 10). The corresponding WS-C `Register` message MUST include the endpoint reference (EPR) of  
2843 the protocol handler of the WS-HumanTask Processor that the WS-HumanTask Parent MUST use to  
2844 send all protocol messages to WS-HumanTask Processor. This EPR is the value contained in the  
2845 `ParticipantProtocolService` element of the `Register` message. Furthermore, the registration  
2846 MUST be as a `HumanTask` participant by specifying the corresponding value in the  
2847 `ProtocolIdentifier` element of the `Register` message. The WS-HumanTask Parent reacts to that  
2848 message by sending back a `RegisterResponse` message. This message MUST contain in its  
2849 `CoordinatorProtocolService` element the EPR of the protocol handler of the parent application,  
2850 which MUST be used by the WS-HumanTask Processor for sending protocol messages to the parent  
2851 application (step (3) in Figure 10).

2852 Now the instance of the human task is activated by the WS-HumanTask Processor, so the assigned  
2853 person can perform the task (e.g. the risk assessment). Once the human task is successfully completed,  
2854 a response message MUST be passed back to the parent application (step (4a) in Figure 10) by WS-  
2855 HumanTask Processor.

2856 **Scenario 2:** If the human task is not completed with a result, but the assigned person determines that the  
2857 task can be skipped (and hence reaches its *Obsolete* final state), then a “skipped” coordination protocol  
2858 message MUST be sent from the WS-HumanTask Processor to its parent application (step (4b) in Figure  
2859 10). No response message is passed back.

2860 **Scenario 3:** If the WS-HumanTask Parent needs to end prematurely before the invoked human task has  
2861 been completed, it MUST send an `exit` coordination protocol message to the WS-HumanTask  
2862 Processor causing the WS-HumanTask Processor to end its processing. A response message SHOULD  
2863 NOT be passed back by WS-HumanTask Processor.

2864 In case of notifications to WS-HumanTask Processor, only some of the overriding attributes are  
2865 propagated with the request message. Only priority and people assignments MAY be overridden for a  
2866 notification, and the elements `isSkipable`, `expirationTime` and `attachments` MUST be ignored if present by  
2867 WS-HumanTask Processor. Likewise, the WS-HumanTask coordination context, `attachments` and the  
2868 callback EPR do not apply to notifications and MUST be ignored as well by WS-HumanTask Processor.  
2869 Finally, a notification SHOULD NOT return WS-HumanTask coordination protocol messages. There  
2870 SHOULD NOT be a message exchange beyond the initiating request message between the WS-  
2871 HumanTask Processor and WS-HumanTask Parent.

2872 **8.1 Human Task Coordination Protocol Messages**

2873 The following section describes the behavior of the human task with respect to the protocol messages  
 2874 exchanged with its requesting application which is human task aware. In particular, we describe which  
 2875 state transitions trigger which protocol message and vice versa. WS-HumanTask Parent MUST support  
 2876 WS-HumanTask Coordination protocol messages in addition to application requesting, responding and  
 2877 fault messages.

2878 See diagram in section 4.10 “Human Task Behavior and State Transitions”.

- 2879 1. The initiating message containing a WS-HumanTask coordination context is received by the WS-  
 2880 HumanTask Processor. This message MAY include ad hoc attachments that are to be made  
 2881 available to the WS-HumanTask Processor. A new task is created. As part of the context, an EPR  
 2882 of the registration service MUST be passed by WS-HumanTask Parent. This registration service  
 2883 MUST be used by the hosting WS-HumanTask Processor to register the protocol handler  
 2884 receiving the WS-HumanTask protocol messages sent by the requesting Application. If an error  
 2885 occurs during the task instantiation the final state *Error* is reached and protocol message *fault*  
 2886 MUST be sent to the requesting application by WS-HumanTask Processor.
- 2887 2. On successful completion of the task an application level response message MUST be sent and  
 2888 the task moved to state *Completed*. When this happens, attachments created during the  
 2889 processing of the task MAY be added to the response message. Attachments that had been  
 2890 passed in the initiating message MUST NOT be returned. The response message outcome  
 2891 MUST be set to the outcome of the task.
- 2892 3. On unsuccessful completion (completion with a fault message), an application level fault  
 2893 message MUST be sent and the task moved to state *Failed*. When this happens, attachments  
 2894 created during the processing of the task MAY be added to the response message. Attachments  
 2895 that had been passed in the initiating message MUST NOT be returned.
- 2896 4. If the task experiences a non-recoverable error protocol message *fault* MUST be sent and  
 2897 the task moved to state *Error*. Attachments MUST NOT be returned.
- 2898 5. If the task is skipable and is skipped then the WS-HumanTask Processor MUST send the  
 2899 protocol message *skipped* and task MUST be moved to state *Obsolete*. Attachments MUST  
 2900 NOT be returned.
- 2901 6. On receipt of protocol message *exit* the task MUST be moved to state *Exited*. This indicates  
 2902 that the requesting application is no longer interested in any result produced by the task.

2903 The following table summarizes this behavior, the messages sent, and their direction, i.e., whether a  
 2904 message is sent from the requesting application to the task (“out” in the column titled Direction) or vice  
 2905 versa (“in”).

2906

| 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                                      |

## 2907 8.2 Protocol Messages

2908 All WS-HumanTask protocol messages have the following type:

```
2909 <xsd:complexType name="tProtocolMsgType">
2910   <xsd:sequence>
2911     <xsd:any namespace="##other" processContents="lax"
2912       minOccurs="0" maxOccurs="unbounded" />
2913   </xsd:sequence>
2914   <xsd:anyAttribute namespace="##other" processContents="lax" />
2915 </xsd:complexType>
```

2916 This message type is extensible and any implementation MAY use this extension mechanism to define  
2917 proprietary attributes and content which are out of the scope of this specification.

### 2918 8.2.1 Protocol Messages Received by a Task Parent

2919 The following is the definition of the `htcp:skipped` message.

```
2920 <xsd:element name="skipped" type="htcp:tProtocolMsgType" />
2921 <wsdl:message name="skipped">
2922   <wsdl:part name="parameters" element="htcp:skipped" />
2923 </wsdl:message>
```

2924 The `htcp:skipped` message is used to inform the task parent (i.e. the requesting application) that the  
2925 invoked task has been skipped. The task does not return any result.

2926 The following is the definition of the `htcp:fault` message.

```
2927 <xsd:element name="fault" type="htcp:tProtocolMsgType" />
2928 <wsdl:message name="fault">
2929   <wsdl:part name="parameters" element="htcp:fault" />
2930 </wsdl:message>
```

2931 The `htcp:fault` message is used to inform the task parent that the task has ended abnormally. The  
2932 task does not return any result.

### 2933 8.2.2 Protocol Messages Received by a Task

2934 Upon receipt of the following `htcp:exit` message the task parent informs the task that it is no longer  
2935 interested in its results.

```
2936 <xsd:element name="exit" type="htcp:tProtocolMsgType" />
2937 <wsdl:message name="exit">
2938   <wsdl:part name="parameters" element="htcp:exit" />
2939 </wsdl:message>
```

## 2940 8.3 WSDL of the Protocol Endpoints

2941 Protocol messages are received by protocol participants via operations of dedicated ports called protocol  
2942 endpoints. In this section we specify the WSDL port types of the protocol endpoints needed to run the  
2943 WS-HumanTask coordination protocol.

### 2944 8.3.1 Protocol Endpoint of the Task Parent

2945 An application that wants to create a task and wants to become a task parent MUST provide an endpoint  
2946 implementing the following port type. This endpoint is the protocol endpoint of the task parent receiving  
2947 protocol messages of the WS-HumanTask coordination protocol from a task. The operation used by the  
2948 task to send a certain protocol message to the task parent is named by the message name of the protocol  
2949 message concatenated by the string `Operation`. For example, the `skipped` message MUST be passed  
2950 to the task parent by using the operation named `skippedOperation`.

```
2951 <wsdl:portType name="clientParticipantPortType">
```

```

2952 <wsdl:operation name="skippedOperation">
2953   <wsdl:input message="htcp:skipped" />
2954 </wsdl:operation>
2955 <wsdl:operation name="faultOperation">
2956   <wsdl:input message="htcp:fault" />
2957 </wsdl:operation>
2958 </wsdl:portType>

```

### 2959 8.3.2 Protocol Endpoint of the Task

2960 For a WS-HumanTask Definition a task MUST provide an endpoint implementing the following port type.  
 2961 This endpoint is the protocol endpoint of the task receiving protocol messages of the WS-HumanTask  
 2962 coordination protocol from a task parent. The operation used by the task parent to send a certain protocol  
 2963 message to a task is named by the message name of the protocol message concatenated by the string  
 2964 Operation. For example, the `exit` protocol message MUST be passed to the task by using the  
 2965 operation named `exitOperation`.

```

2966 <wsdl:portType name="humanTaskParticipantPortType">
2967   <wsdl:operation name="exitOperation">
2968     <wsdl:input message="htcp:exit" />
2969   </wsdl:operation>
2970 </wsdl:portType>

```

### 2971 8.4 Providing Human Task Context

2972 The task context information is exchanged between the requesting application and a task or a notification.  
 2973 In case of tasks, this information is passed as header fields of the request and response messages of the  
 2974 task's operation. In case of notifications, this information is passed as header fields of the request  
 2975 message of the notification's operation.

#### 2976 8.4.1 SOAP Binding of Human Task Context

2977 In general, a SOAP binding specifies for message header fields how they are bound to SOAP headers. In  
 2978 case of WS-HumanTask, the `humanTaskRequestContext` and `humanTaskResponseContext`  
 2979 elements are simply mapped to SOAP header as a whole. The following listings show the SOAP binding  
 2980 of the human task request context and human task response context in an infoset representation.

```

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:humanTaskRequestContext>
2986       <htc:priority>...</htc:priority>?
2987       <htc:attachments>...</htc:attachments>?
2988       <htc:peopleAssignments>...</htc:peopleAssignments>?
2989       <htc:isSkipable>...</htc:isSkipable>?
2990       <htc:activationDeferralTime>...</htc:activationDeferralTime>?
2991       <htc:expirationTime>...</htc:expirationTime>?
2992       ... extension elements ...
2993     </htc:humanTaskRequestContext>
2994   </S:Header>
2995   <S:Body>
2996     ...
2997   </S:Body>
2998 </S:Envelope>

```

```

2999
3000 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"

```

```

3001         xmlns:htc="http://docs.oasis-open.org/ns/bpel4people/ws-
3002 humantask/context/200803">
3003     <S:Header>
3004         <htc:humanTaskResponseContext>
3005             <htc:priority>...</htc:priority>?
3006             <htc:attachments>...</htc:attachments>?
3007             <htc:actualOwner>...</htc:actualOwner>?
3008             <htc:actualPeopleAssignments>...</htc:actualPeopleAssignments>?
3009             <htc:outcome>...</htc:outcome>?
3010             ... extension elements ...
3011         </htc:humanTaskResponseContext>
3012     </S:Header>
3013     <S:Body>
3014         ...
3015     </S:Body>
3016 </S:Envelope>

```

3017 The following listing is an example of a SOAP message containing a human task request context.

```

3018 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
3019           xmlns:htc="http://docs.oasis-open.org/ns/bpel4people/ws-
3020 humantask/context/200803">
3021     <S:Header>
3022         <htc:humanTaskRequestContext>
3023             <htc:priority>0</htc:priority>
3024             <htc:peopleAssignments>
3025                 <htc:potentialOwners>
3026                     <htt:organizationalEntity>
3027                         <htt:user>Alan</htt:user>
3028                         <htt:user>Dieter</htt:user>
3029                         <htt:user>Frank</htt:user>
3030                         <htt:user>Gerhard</htt:user>
3031                         <htt:user>Ivana</htt:user>
3032                         <htt:user>Karsten</htt:user>
3033                         <htt:user>Matthias</htt:user>
3034                         <htt:user>Patrick</htt:user>
3035                     </htt:organizationalEntity>
3036                 </htc:potentialOwners>
3037             </htc:peopleAssignments>
3038         </htc:humanTaskRequestContext>
3039     </S:Header>
3040     <S:Body>...</S:Body>
3041 </S:Envelope>

```

## 3042 8.4.2 Overriding Task Definition People Assignments

3043 The task context information exchanged contains a `potentialOwners` element, which can be used at  
3044 task creation time to override the set of task assignments that we defined in the original task definition.  
3045 Compliant implementations MUST allow overriding of simple tasks and routing patterns that are a single-  
3046 level deep, i.e. routing patterns that don't have nested routing patterns. If the task context  
3047 `potentialOwners` contains a list of `htt:user` and `htt:group`, and the task definition contains a  
3048 routing pattern element `htt:parallel` or `htt:sequence` that has as its only children `htt:user` and  
3049 `htt:group` elements, the WS-HumanTask Processor MUST replace the list in the task definition with the  
3050 list in the task context. If the task definition contains only a list of `htt:user` and `htt:group`, then the  
3051 WS-HumanTask Processor MUST replace the list of users from the task definition with the list of users in  
3052 the task context.

## 3053 8.5 Human Task Policy Assertion

3054 In order to support discovery of Web services that support the human task contract that are available for  
3055 coordination by another service, a *human task policy* assertion is defined by WS-HumanTask. This policy  
3056 assertion can be associated with the business operation used by the invoking component (recall that the  
3057 human task is restricted to have exactly one business operation). In doing so, the provider of a human  
3058 task can signal whether or not the corresponding task can communicate with an invoking component via  
3059 the WS-HumanTask coordination protocol.

3060 The following describes the policy assertion used to specify that an operation can be used to instantiate a  
3061 human task with the proper protocol in place:

```
3062 <http:HumanTaskAssertion wsp:Optional="true"? ...>  
3063   ...  
3064 </http:HumanTaskAssertion>
```

3065 /http:HumanTaskAssertion

3066 This policy assertion specifies that the WS-HumanTask Parent, in this case the sender, **MUST**  
3067 include context information for a human task coordination type passed with the message. The  
3068 receiving human task **MUST** be instantiated with the WS-Human Task protocol in place by the  
3069 WS-HumanTask Processor.

3070 /http:HumanTaskAssertion/@wsp:Optional="true"

3071 As defined in WS-Policy [WS-Policy], this is the compact notation for two policy alternatives, one  
3072 with and one without the assertion. Presence of both policy alternatives indicates that the  
3073 behavior indicated by the assertion is optional, such that a WS-HumanTask coordination context  
3074 **MAY** be passed with an input message. If the context is passed the receiving human task **MUST**  
3075 be instantiated with the WS-HumanTask protocol in place. The absence of the assertion is  
3076 interpreted to mean that a WS-HumanTask coordination context **SHOULD NOT** be passed with  
3077 an input message.

3078 The human task policy assertion indicates behavior for a single operation, thus the assertion has an  
3079 Operation Policy Subject. WS-PolicyAttachment [WS-PolAtt] defines two policy attachment points with  
3080 Operation Policy Subject, namely wsdl:portType/wsdl:operation and wsdl:binding/wsdl:operation.

3081 The `<http:HumanTaskAssertion>` policy assertion can also be used for notifications. In that case it  
3082 means that the WS-HumanTask Parent, in this case the sender, **MAY** pass the human task context  
3083 information with the message. Other headers, including headers with the coordination context are  
3084 ignored.

---

## 3085 9 Task Parent Interactions with Lean Tasks

### 3086 9.1 Operations for Task Parent Applications

3087 A number of operations are involved in the life cycle of a lean task definition. These comprise:

- 3088 • Registering a lean task definition, such that it is available for later use
- 3089 • Unregistering a lean task definition, such that it is no longer available for later use
- 3090 • Listing lean task definitions, to determine what is available for use
- 3091 • Creating a lean task from a lean task definition

3092 An operation takes a well-defined set of parameters as its input. Passing an illegal parameter or an illegal  
3093 number of parameters **MUST** result in the `htlt:illegalArgumentFault` being returned. Invoking an  
3094 operation that is not allowed in the current state of the lean task definition **MUST** result in an  
3095 `htlt:illegalStateFault`.

3096 By default, the identity of the person on behalf of which the operation is invoked is passed to the WS-  
3097 HumanTask Processor. When the person is not authorized to perform the operation the  
3098 `htlt:illegalAccessFault` **MUST** be returned.

3099 This specification does not stipulate the authentication, addressing, and binding scheme employed when  
3100 calling an operation. This can be achieved using different mechanisms (e.g. WS-Security, WS-  
3101 Addressing).

### 3102 9.2 Lean Task Interactions

3103 To enable lightweight task definition and creation by a WS-HumanTask Parent, a conformant WS-  
3104 HumanTask Processor **MUST** provide the following operations:

- 3105 • `registerLeanTaskDefinition` API for registration
- 3106 • `unregisterLeanTaskDefinition` API for retraction
- 3107 • `listLeanTaskDefinitions` API for enumeration
- 3108 • `createLeanTask` and `createLeanTaskAsync` APIs for creation

3109 and invoke the following callback operation in response to `createLeanTaskAsync`:

- 3110 • `createLeanTaskAsyncCallback`

#### 3111 9.2.1 Register a Lean Task Definition

```
3112 <xsd:element name="registerLeanTaskDefinition">
3113   <xsd:complexType>
3114     <xsd:sequence>
3115       <xsd:element name="taskDefinition" type="htd:tLeanTask" />
3116     </xsd:sequence>
3117   </xsd:complexType>
3118 </xsd:element>
3119 <xsd:element name="registerLeanTaskDefinitionResponse">
3120   <xsd:complexType>
3121     <xsd:sequence>
3122       <xsd:element name="taskName" type="xsd:NCName" />
3123     </xsd:sequence>
3124   </xsd:complexType>
3125 </xsd:element>
```

3126 The `htlt:registerLeanTaskDefinition` operation is used to create a new Lean Task definition that  
3127 is available for future listing and consumption by the `htlt:listLeanTaskDefinitions` and  
3128 `htlt:createLeanTask/htlt:createLeanTaskAsync` operations. If an existing Lean Task exists at  
3129 the same name as the `htd:tLeanTask/@Name`, the WSHumanTask Processor SHOULD return an  
3130 `htlt:illegalStateFault`.

## 3131 9.2.2 Unregister a Lean Task Definition

```
3132 <xsd:element name="unregisterLeanTaskDefinition">  
3133   <xsd:complexType>  
3134     <xsd:sequence>  
3135       <xsd:element name="taskName" type="xsd:NCName" />  
3136     </xsd:sequence>  
3137   </xsd:complexType>  
3138 </xsd:element>  
3139 <xsd:element name="unregisterLeanTaskDefinitionResponse">  
3140   <xsd:complexType>  
3141     <xsd:sequence>  
3142       <xsd:element name="taskName" type="xsd:NCName" />  
3143     </xsd:sequence>  
3144   </xsd:complexType>  
3145 </xsd:element>
```

3146 The `htlt:unregisterLeanTaskDefinition` operation is used to remove a Lean Task available for  
3147 future listing and consumption by the `htlt:listLeanTaskDefinitions` and  
3148 `htlt:createLeanTask/htlt:createLeanTaskAsync` operations. The WS-HumanTask Processor  
3149 SHOULD also move any instances of lean tasks of this task definition to "Error" state. If the Lean Task  
3150 does not already exist as a registered element, the WS-HumanTask Processor MUST return an  
3151 `htlt:illegalArgumentFault`.

## 3152 9.2.3 List Lean Task Definitions

```
3153 <xsd:element name="listLeanTaskDefinitions">  
3154   <xsd:complexType>  
3155     <xsd:sequence>  
3156       <xsd:annotation>  
3157         <xsd:documentation>Empty message</xsd:documentation>  
3158       </xsd:annotation>  
3159     </xsd:sequence>  
3160   </xsd:complexType>  
3161 </xsd:element>  
3162 <xsd:element name="listLeanTaskDefinitionsResponse">  
3163   <xsd:complexType>  
3164     <xsd:sequence>  
3165       <xsd:element name="leanTaskDefinitions">  
3166         <xsd:complexType>  
3167           <xsd:sequence>  
3168             <xsd:element name="leanTaskDefinition" type="htd:tLeanTask"  
3169 minOccurs="0" maxOccurs="unbounded" />  
3170           </xsd:sequence>  
3171         </xsd:complexType>  
3172       </xsd:element>  
3173     </xsd:sequence>  
3174   </xsd:complexType>  
3175 </xsd:element>
```

3176 The `htlt:listLeanTaskDefinitions` operation is used to query the list of `htd:tLeanTask`  
3177 elements that are registered Lean Tasks, as registered by the `htlt:registerLeanTaskDefinition`  
3178 operation, and not subsequently unregistered by `htlt:unregisterLeanTaskDefinition`.

## 3179 9.2.4 Create a Lean Task

```
3180 <xsd:element name="CreateLeanTask">  
3181   <xsd:complexType>  
3182     <xsd:sequence>  
3183       <xsd:element name="inputMessage">  
3184         <xsd:complexType>  
3185           <xsd:sequence>  
3186             <xsd:any processContents="lax" namespace="##any" />  
3187           </xsd:sequence>  
3188         </xsd:complexType>  
3189       </xsd:element>  
3190       <xsd:element name="taskDefinition" type="htd:tLeanTask" minOccurs="0"/>  
3191       <xsd:element name="taskName" type="xsd:NCName" minOccurs="0" />  
3192     </xsd:sequence>  
3193   </xsd:complexType>  
3194 </xsd:element>  
3195 <xsd:element name="CreateLeanTaskResponse">  
3196   <xsd:complexType>  
3197     <xsd:sequence>  
3198       <xsd:element name="outputMessage">  
3199         <xsd:complexType>  
3200           <xsd:sequence>  
3201             <xsd:any processContents="lax" namespace="##any" />  
3202           </xsd:sequence>  
3203         </xsd:complexType>  
3204       </xsd:element>  
3205     </xsd:sequence>  
3206   </xsd:complexType>  
3207 </xsd:element>  
3208
```

3209 The `htlt:createLeanTask` operation is called by a WS-HumanTask Parent to create a task based on  
3210 a Lean Task definition. This task definition either can be passed in directly to the operation or can  
3211 reference a Lean Task definition previously sent via `htlt:registerLeanTaskDefinition`. These  
3212 tasks follow the standard pattern of the Human Task Coordination protocol and is the operation on the  
3213 portType used to create a task in that standard pattern, using the `humanTaskRequestContext` and  
3214 `humanTaskResponseContext` as described in section 8.4.

3215 If both `taskName` and `taskDefinition` are set, the WS-HumanTask Processor MUST return an  
3216 `htlt:illegalArgumentFault`. If `taskName` is set and a lean task has been registered by that name,  
3217 the WS-HumanTask Process MUST use the registered lean task definition to create the task. If `taskName`  
3218 is not set and a lean task has not been registered by that name, the WS-HumanTask Processor MUST  
3219 return an `htlt:illegalArgumentFault`. If `taskDefinition` is set, the WS-HumanTask Processor MUST  
3220 use the `taskDefinition` element as the type of the task to create. The WS-HumanTask Processor MUST  
3221 use the `inputMessage` as the input message of the task and return the output message of the task in  
3222 the `outputMessage` element.

3223 The `htlt:createLeanTask` operation is long-running because its execution includes the user  
3224 interaction with the task owner. As a result, it is not meaningful to bind the request-response operation to  
3225 a protocol that blocks any resources until the response is returned.

3226 Alternatively, instead of invoking the long-running request-response operation defined above, an  
3227 interaction style using an asynchronous callback operation can be used. In this case, the WS-HumanTask  
3228 Parent invokes the following `htlt:createLeanTaskAsync` operation and, as described in section 10,

3229 passes a WS-Addressing endpoint reference (EPR) in order to provide a callback address for delivering  
3230 the lean task's output.

3231 Technically, `htlt:createLeanTaskAsync` is also a request-response operation in order to enable  
3232 returning faults, but it returns immediately to the caller if the lean task is created successfully, without  
3233 waiting for the lean task to complete.

```
3234 <xsd:element name="createLeanTaskAsync">  
3235   <xsd:complexType>  
3236     <xsd:sequence>  
3237       <xsd:element name="inputMessage">  
3238         <xsd:complexType>  
3239           <xsd:sequence>  
3240             <xsd:any processContents="lax" namespace="##any" />  
3241           </xsd:sequence>  
3242         </xsd:complexType>  
3243       </xsd:element>  
3244       <xsd:element name="taskDefinition" type="htd:tLeanTask" minOccurs="0"/>  
3245       <xsd:element name="taskName" type="xsd:NCName" minOccurs="0" />  
3246     </xsd:sequence>  
3247   </xsd:complexType>  
3248 </xsd:element>  
3249 <xsd:element name="createLeanTaskAsyncResponse">  
3250   <xsd:complexType>  
3251     <xsd:sequence/>  
3252   </xsd:complexType>  
3253 </xsd:element>
```

3254 Upon completion of the lean task, the WS-HumanTask Processor invokes the callback operation  
3255 `htlt:createLeanTaskAsyncCallback` at the callback address specified in the EPR passed by the  
3256 WS-HumanTask Parent.

```
3257 <xsd:element name="createLeanTaskAsyncCallback">  
3258   <xsd:complexType>  
3259     <xsd:sequence>  
3260       <xsd:element name="outputMessage">  
3261         <xsd:complexType>  
3262           <xsd:sequence>  
3263             <xsd:any processContents="lax" namespace="##any" />  
3264           </xsd:sequence>  
3265         </xsd:complexType>  
3266       </xsd:element>  
3267     </xsd:sequence>  
3268   </xsd:complexType>  
3269 </xsd:element>
```

## 3270 9.2.5 Endpoints for Lean Task Operations

3271 A WS-HumanTask Processor MUST provide an endpoint implementing the following port type. This  
3272 endpoint is used to register, unregister, and list lean task definitions, and create a lean task given a  
3273 particular definition and input message.

```
3274 <wsdl:portType name="leanTaskOperations">  
3275   <wsdl:operation name="registerLeanTaskDefinition">  
3276     <wsdl:input message="registerLeanTaskDefinition"/>  
3277     <wsdl:output message="registerLeanTaskDefinitionResponse"/>  
3278     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>  
3279     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>  
3280   </wsdl:operation>  
3281 </wsdl:portType>  
3282
```

```

3283 <wsdl:operation name="unregisterLeanTaskDefinition">
3284   <wsdl:input message="unregisterLeanTaskDefinition"/>
3285   <wsdl:output message="unregisterLeanTaskDefinitionResponse"/>
3286   <wsdl:fault name="illegalArgumentFault" message="illegalArgumentFault"/>
3287   <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
3288 </wsdl:operation>
3289
3290 <wsdl:operation name="listLeanTaskDefinitions">
3291   <wsdl:input message="listLeanTaskDefinitions"/>
3292   <wsdl:output message="listLeanTaskDefinitionsResponse"/>
3293   <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
3294 </wsdl:operation>
3295
3296 <wsdl:operation name="createLeanTask">
3297   <wsdl:input message="createLeanTask"/>
3298   <wsdl:output message="createLeanTaskResponse"/>
3299   <wsdl:fault name="illegalArgumentFault" message="illegalArgumentFault"/>
3300   <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
3301 </wsdl:operation>
3302
3303 <wsdl:operation name="createLeanTaskAsync">
3304   <wsdl:input message="createLeanTaskAsync"/>
3305   <wsdl:output message="createLeanTaskAsyncResponse"/>
3306   <wsdl:fault name="illegalArgumentFault" message="illegalArgumentFault"/>
3307   <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
3308 </wsdl:operation>
3309
3310 </wsdl:portType>

```

3311 A WS-HumanTask Parent invoking the `htlt:createLeanTaskAsync` operation MUST provide an  
3312 endpoint implementing the following callback port type.

```

3313 <wsdl:portType name="leanTaskCallbackOperations">
3314
3315   <wsdl:operation name="createLeanTaskAsyncCallback">
3316     <wsdl:input message="createLeanTaskAsyncCallback"/>
3317   </wsdl:operation>
3318
3319 </wsdl:portType>

```

3320

## 10 Providing Callback Information for Human Tasks

3321 WS-HumanTask extends the information model of a WS-Addressing endpoint reference (EPR) defined in  
3322 [WS-Addr-Core] (see [WS-Addr-SOAP] and [WS-Addr-WSDL] for more details). This extension is needed  
3323 to support passing information to human tasks about ports and operations of a caller receiving responses  
3324 from such human tasks.

3325 Passing this callback information from a WS-HumanTask Parent (i.e. a requesting application) to the WS-  
3326 HumanTask Processor MAY override static deployment information that may have been set.

### 10.1 EPR Information Model Extension

3327 Besides the properties of an endpoint reference (EPR) defined by [WS-Addr-Core] WS-HumanTask  
3328 defines the following abstract properties:

3330 [response action] : xsd:anyURI (0..1)

3331 This property contains the value of the [action] message addressing property to be sent within the  
3332 response message.

3333 [response operation] : xsd:NCName (0..1)

3334 This property contains the name of a WSDL operation.

3335 Each of these properties is a child element of the [metadata] property of an endpoint reference. An  
3336 endpoint reference passed by a caller to a WS-HumanTask Processor MUST contain the [metadata]  
3337 property. Furthermore, this [metadata] property MUST contain either a [response action] property or a  
3338 [response operation] property.

3339 If present, the value of the [response action] property MUST be used by the WS-HumanTask Processor  
3340 hosting the responding human task to specify the value of the [action] message addressing property of  
3341 the response message sent back to the caller. Furthermore, the [destination] property of this response  
3342 message MUST be copied from the [address] property of the EPR contained in the original request  
3343 message by the WS-HumanTask Processor.

3344 If present, the value of the [response operation] property MUST be the name of an operation of the port  
3345 type implemented by the endpoint denoted by the [address] property of the EPR. The corresponding port  
3346 type MUST be included as a WSDL 1.1 definition nested within the [metadata] property of the EPR (see  
3347 [WS-Addr-WSDL]). The WS-HumanTask Processor hosting the responding human task MUST use the  
3348 value of the [response operation] property as operation of the specified port type at the specified endpoint  
3349 to send the response message. Furthermore, the [metadata] property MUST contain WSDL 1.1 binding  
3350 information corresponding to the port type implemented by the endpoint denoted by the [address]  
3351 property of the EPR.

3352 The EPR sent from the caller to the WS-HumanTask Processor MUST identify the instance of the caller.  
3353 This MUST be done by the caller in one of the two ways: First, the value of the [address] property can  
3354 contain a URL with appropriate parameters uniquely identifying the caller instance. Second, appropriate  
3355 [reference parameters] properties are specified within the EPR. The values of these [reference  
3356 parameters] uniquely identify the caller within the scope of the URI passed within the [address] property.

### 10.2 XML Infoset Representation

3357 The following describes the infoset representation of the EPR extensions introduced by WS-HumanTask:

```
3359 <wsa:EndpointReference>  
3360   <wsa:Address>xsd:anyURI</wsa:Address>  
3361   <wsa:ReferenceParameters>xsd:any*</wsa:ReferenceParameters>?  
3362   <wsa:Metadata>  
3363     <htcp:responseAction>xsd:anyURI</htcp:responseAction>?  
3364     <htcp:responseOperation>xsd:NCName</htcp:responseOperation>?  
3365   </wsa:Metadata>
```

3366 `</wsa:EndpointReference>`

3367 `/wsa:EndpointReference/wsa:Metadata`

3368 This element of the EPR MUST be sent by WS-HumanTask Parent, the caller, to the WS-  
 3369 HumanTask Processor . It MUST either contain WSDL 1.1 metadata specifying the information to  
 3370 access the endpoint (i.e. its port type, bindings or ports) according to [WS-Addr-WSDL] as well as  
 3371 a `<htcp:responseOperation>` element, or it MUST contain a `<htcp:responseAction>`  
 3372 element.

3373 `/wsa:EndpointReference/wsa:Metadata/htcp:responseAction`

3374 This element (of type `xsd:anyURI`) specifies the value of the [action] message addressing  
 3375 property to be used by the receiving WS-HumanTask Processor when sending the response  
 3376 message from the WS-HumanTask Processor back to the caller. If this element is specified the  
 3377 `<htcp:responseOperation>` element MUST NOT be specified by the caller.

3378 `/wsa:EndpointReference/wsa:Metadata/htcp:responseOperation`

3379 This element (of type `xsd:NCName`) specifies the name of the operation that MUST be used by  
 3380 the receiving WS-HumanTask Processor to send the response message from the WS-  
 3381 HumanTask Processor back to the caller.. If this element is specified the  
 3382 `<htcp:responseAction>` element MUST NOT be specified by the WS-HumanTask Parent.

3383 Effectively, WS-HumanTask defines two ways to pass callback information from the caller to the human  
 3384 task. First, the EPR contains just the value of the [action] message addressing property that MUST be  
 3385 used by the WS-HumanTask Processor within the response message (i.e. the  
 3386 `<htcp:responseAction>` element). Second, the EPR contains the WSDL 1.1 metadata for the port  
 3387 receiving the response operation. In this case, for the callback information the WS-HumanTask Parent  
 3388 MUST specify which operation of that port is to be used (i.e. the `<htcp:responseOperation>`  
 3389 element). In both cases, the response is typically sent to the address specified in the `<wsa:Address>`  
 3390 element of the EPR contained in the original request message; note, that [WS-Addr-WSDL] does not  
 3391 exclude redirection to other addresses than the one specified, but the corresponding mechanisms are out  
 3392 of the scope of the specification.

3393 The following example of an endpoint reference shows the usage of the `<htcp:responseAction>`  
 3394 element. The `<wsa:Metadata>` elements contain the `<htcp:responseAction>` element that  
 3395 specifies the value of the [action] message addressing property to be used by the WS-HumanTask  
 3396 Processor when sending the response message back to the caller. This value is  
 3397 `http://example.com/LoanApproval/approvalResponse`. The value of the [destination] message  
 3398 addressing property to be used is given in the `<wsa:Address>` element, namely  
 3399 `http://example.com/LoanApproval/loan?ID=42`. Note that this URL includes the HTTP search  
 3400 part with the parameter `ID=42` which uniquely identifies the instance of the caller.

```

3401 <wsa:EndpointReference
3402   xmlns:wsa="http://www.w3.org/2005/08/addressing">
3403
3404   <wsa:Address>http://example.com/LoanApproval/loan?ID=42</wsa:Address>
3405
3406   <wsa:Metadata>
3407     <htcp:responseAction>
3408       http://example.com/LoanApproval/approvalResponse
3409     </htcp:responseAction>
3410   </wsa:Metadata>
3411
3412 </wsa:EndpointReference>
  
```

3413 The following example of an endpoint reference shows the usage of the `<htcp:responseOperation>`  
 3414 element and corresponding WSDL 1.1 metadata. The port type of the caller that receives the response  
 3415 message from the WS-HumanTask Processor is defined using the `<wsdl:portType>` element. In our  
 3416 example it is the `LoanApprovalPT` port type. The definition of the port type is nested in a corresponding  
 3417 WSLD 1.1 `<wsdl:definitions>` element in the `<wsa:Metadata>` element. This

3418 <wsdl:definitions> element also contains a binding for this port type as well as a corresponding  
3419 port definition nested in a <wsdl:service> element. The <http:responseOperation> element  
3420 specifies that the approvalResponse operation of the LoanApprovalPT port type is used to send the  
3421 response to the caller. The address of the actual port to be used which implements the  
3422 LoanApprovalPT port type and thus the approvalResponse operation is given in the  
3423 <wsa:Address> element, namely the URL http://example.com/LoanApproval/loan. The  
3424 unique identifier of the instance of the caller is specified in the <xmp:MyInstanceID> element nested in  
3425 the <wsa:ReferenceParameters> element.

```
3426 <wsa:EndpointReference
3427   xmlns:wsa="http://www.w3.org/2005/08/addressing">
3428
3429   <wsa:Address>http://example.com/LoanApproval/loan</wsa:Address>
3430
3431   <wsa:ReferenceParameters>
3432     <xmp:MyInstanceID>42</xmp:MyInstanceID>
3433   </wsa:ReferenceParameters>
3434
3435   <wsa:Metadata>
3436
3437     <wsdl:definitions ...>
3438
3439       <wsdl:portType name="LoanApprovalPT">
3440         <wsdl:operation name="approvalResponse">...</wsdl:operation>
3441         ...
3442       </wsdl:portType>
3443
3444       <wsdl:binding name="LoanApprovalSoap" type="LoanApprovalPT">
3445         ...
3446       </wsdl:binding>
3447
3448       <wsdl:service name="LoanApprovalService">
3449         <wsdl:port name="LA" binding="LoanApprovalSoap">
3450           <soap:address
3451             location="http://example.com/LoanApproval/loan" />
3452         </wsdl:port>
3453         ...
3454       </wsdl:service>
3455
3456     </wsdl:definitions>
3457
3458     <http:responseOperation>approvalResponse</http:responseOperation>
3459
3460   </wsa:Metadata>
3461 </wsa:EndpointReference>
```

### 3463 10.3 Message Addressing Properties

3464 Message addressing properties provide references for the endpoints involved in an interaction at the  
3465 message level. For this case, WS-HumanTask Processor uses the message addressing properties  
3466 defined in [WS-Addr-Core] for the request message as well as for the response message.

3467 The request message sent by the caller (i.e. the requesting application) to the human task uses the  
3468 message addressing properties as described in [WS-Addr-Core]. WS-HumanTask refines the use of the  
3469 following message addressing properties:

- 3470 • The [reply endpoint] message addressing property MUST contain the EPR to be used by the WS-  
3471 HumanTask Processor to send its response to.

3472 Note that the [fault endpoint] property MUST NOT be used by WS-HumanTask Processor. This is  
3473 because via one-way operation no application level faults are returned to the caller.

3474 The response message sent by the WS-HumanTask Processor to the caller uses the message  
3475 addressing properties as defined in [WS-Addr-Core] and refines the use of the following properties:

- 3476 • The value of the [action] message addressing property is set as follows:
  - 3477 • If the original request message contains the `<htcp:responseAction>` element in the  
3478 `<wsa:Metadata>` element of the EPR of the [reply endpoint] message addressing property,  
3479 the value of the former element MUST be copied into the [action] property of the response  
3480 message by WS-HumanTask Processor.
  - 3481 • If the original request message contains the `<htcp:responseOperation>` element (and,  
3482 thus, WSDL 1.1 metadata) in the `<wsa:Metadata>` element of the EPR of the [reply  
3483 endpoint] message addressing property, the value of the [action] message addressing  
3484 property of the response message is determined as follows:
    - 3485 • Assume that the WSDL 1.1 metadata specifies within the binding chosen a value for the  
3486 `soapaction` attribute on the `soap:operation` element of the response operation.  
3487 Then, this value MUST be used as value of the [action] property by WS-HumanTask  
3488 Processor.
    - 3489 • If no such `soapaction` attribute is provided, the value of the [action] property MUST be  
3490 derived as specified in [WS-Addr-WSDL] by WS-HumanTask Processor.
- 3491 • Reference parameters are mapped as specified in [WS-Addr-SOAP].

## 3492 10.4 SOAP Binding

3493 A SOAP binding specifies how abstract message addressing properties are bound to SOAP headers. In  
3494 this case, WS-HumanTask Processor MUST use the mappings as specified by [WS-Addr-SOAP].

3495 The following is an example of a request message sent from the caller to the WS-HumanTask Processor  
3496 containing the `<htcp:responseAction>` element in the incoming EPR. The EPR is mapped to SOAP  
3497 header fields as follows: The endpoint reference to be used by the human task for submitting its response  
3498 message to is contained in the `<wsa:ReplyTo>` element. The address of the endpoint is contained in the  
3499 `<wsa:Address>` element. The identifier of the instance of the caller to be encoded as reference  
3500 parameters in the response message is nested in the `<wsa:ReferenceParameters>` element. The  
3501 value of the `<wsa:Action>` element to be set by the human task in its response to the caller is in the  
3502 `<htcp:responseAction>` element nested in the `<wsa:Metadata>` element of the EPR.

```
3503 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"  
3504   xmlns:wsa="http://www.w3.org/2005/08/addressing"  
3505   xmlns:htcp="http://docs.oasis-open.org/ns/bpel4people/ws-  
3506   humantask/protocol/200803">  
3507  
3508   <S:Header>  
3509     <wsa:ReplyTo>  
3510       <wsa:Address>http://example.com/LoanApproval/loan</wsa:Address>  
3511       <wsa:ReferenceParameters>  
3512         <xmp:MyInstanceID>42</xmp:MyInstanceID>  
3513       </wsa:ReferenceParameters>  
3514       <wsa:Metadata>  
3515         <htcp:responseAction>  
3516           http://example.com/LoanApproval/approvalResponse  
3517         </htcp:responseAction>  
3518       </wsa:Metadata>  
3519     </wsa:ReplyTo>  
3520   </S:Header>  
3521  
3522   <S:Body>...</S:Body>
```

3523 </S:Envelope>

3524 The following is an example of a response message corresponding to the request message discussed  
3525 above. This response is sent from the WS-HumanTask Processor back to the caller. The <wsa:To>  
3526 element contains a copy of the <wsa:Address> element of the original request message. The  
3527 <wsa:Action> element is copied from the <http:responseAction> element of the original request  
3528 message. The reference parameters are copied as standalone elements (the <xmp:MyInstanceID>  
3529 element below) out of the <wsa:ReferenceParameters> element of the request message.

```
3530 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"  
3531   xmlns:wsa="http://www.w3.org/2005/08/addressing">  
3532   <S:Header>  
3533     <wsa:To>  
3534       <wsa:Address>http://example.com/LoanApproval/loan</wsa:Address>  
3535     </wsa:To>  
3536     <wsa:Action>  
3537       http://example.com/LoanApproval/approvalResponse  
3538     </wsa:Action>  
3539     <xmp:MyInstanceID wsa:IsReferenceParameter='true'>  
3540       42  
3541     </xmp:MyInstanceID>  
3542   </S:Header>  
3543   <S:Body>...</S:Body>  
3544 </S:Envelope>
```

3545 The following is an example of a request message sent from the caller to the WS-HumanTask Processor  
3546 containing the <http:responseOperation> element and corresponding WSDL metadata in the  
3547 incoming EPR. The EPR is mapped to SOAP header fields as follows: The endpoint reference to be used  
3548 by the WS-HumanTask Processor for submitting its response message to is contained in the  
3549 <wsa:ReplyTo> element. The address of the endpoint is contained in the <wsa:Address> element.  
3550 The identifier of the instance of the caller to be encoded as reference parameters in the response  
3551 message is nested in the <wsa:ReferenceParameters> element. The WSDL metadata of the  
3552 endpoint is contained in the <wsdl:definitions> element. The name of the operation of the endpoint  
3553 to be used to send the response message to is contained in the <http:responseOperation>  
3554 element. Both elements are nested in the <wsa:Metadata> element of the EPR. These elements  
3555 provide the basis to determine the value of the action header field to be set by the WS-HumanTask  
3556 Processor in its response to the caller.

```
3557 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"  
3558   xmlns:wsa="http://www.w3.org/2005/08/addressing"  
3559   xmlns:http="http://docs.oasis-open.org/ns/bpel4people/ws-  
3560   humantask/protocol/200803">  
3561   <S:Header>  
3562     <wsa:ReplyTo>  
3563       <wsa:Address>http://example.com/LoanApproval/loan</wsa:Address>  
3564     </wsa:ReplyTo>  
3565     <wsa:ReferenceParameters>  
3566       <xmp:MyInstanceID>42</xmp:MyInstanceID>  
3567     </wsa:ReferenceParameters>  
3568     <wsa:Metadata>  
3569       <wsdl:definitions  
3570         targetNamespace="http://example.com/loanApproval"  
3571         xmlns:wsdl="..." xmlns:soap="...">  
3572         <wsdl:portType name="LoanApprovalPT">  
3573           <wsdl:operation name="approvalResponse">  
3574             <wsdl:input name="approvalInput" ... />  
3575         </wsdl:portType>  
3576       </wsdl:definitions>  
3577     </wsa:Metadata>  
3578   </S:Header>
```

```

3579     </wsdl:operation>
3580     ...
3581 </wsdl:portType>
3582
3583 <wsdl:binding name="LoanApprovalSoap"
3584     type="LoanApprovalPT">
3585     ...
3586 </wsdl:binding>
3587
3588 <wsdl:service name="LoanApprovalService">
3589     <wsdl:port name="LA" binding="LoanApprovalSoap">
3590         <soap:address
3591             location="http://example.com/LoanApproval/loan" />
3592     </wsdl:port>
3593     ...
3594 </wsdl:service>
3595 </wsdl:definitions>
3596
3597 <htcp:responseOperation>
3598     approvalResponse
3599 </htcp:responseOperation>
3600
3601 </wsa:Metadata>
3602 </wsa:ReplyTo>
3603
3604 </S:Header>
3605 <S:Body>...</S:Body>
3606 </S:Envelope>

```

3607 The following is an example of a response message corresponding to the request message before; this  
 3608 response is sent from the WS-HumanTask Processor back to the caller. The `<wsa:To>` element contains  
 3609 a copy of the `<wsa:Address>` field of the original request message. The reference parameters are  
 3610 copied as standalone element (the `<xmp:MyInstanceID>` element below) out of the  
 3611 `<htcp:ReferenceParameters>` element of the request message. The value of the `<wsa:Action>`  
 3612 element is composed according to [WS-Addr-WSDL] from the target namespace, port type name, name  
 3613 of the response operation to be used, and name of the input message of this operation given in the code  
 3614 snippet above.

```

3615 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
3616     xmlns:wsa="http://www.w3.org/2005/08/addressing"
3617     xmlns:htd="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803">
3618 <S:Header>
3619     <wsa:To>http://example.com/LoanApproval/loan</wsa:To>
3620     <wsa:Action>
3621         http://example.com/loanApproval/...
3622         ...LoanApprovalPT/approvalResponse/ApprovalInput
3623     </wsa:Action>
3624     <xmp:MyInstanceID wsa:IsReferenceParameter='true'>
3625         42
3626     </xmp:MyInstanceID>
3627 </S:Header>
3628 <S:Body>...</S:Body>
3629 </S:Envelope>

```

---

3630

## 11 Security Considerations

3631 WS-HumanTask does not mandate the use of any specific mechanism or technology for client  
3632 authentication. However, a client **MUST** provide a principal or the principal **MUST** be obtainable by the  
3633 WS-HumanTask Processor.

3634 When using task APIs via SOAP bindings, compliance with the WS-I Basic Security Profile 1.0 is  
3635 **RECOMMENDED**.

---

3636 **12 Conformance**

3637

3638 The XML schema pointed to by the RDDL document at the namespace URI, defined by this specification,  
3639 are considered to be authoritative and take precedence over the XML schema defined in the appendix of  
3640 this document.

3641

3642 There are four conformance targets defined as part of this specification: a WS-HumanTask Definition, a  
3643 WS-HumanTask Processor, a WS-HumanTask Parent and a WS-HumanTask Client (see section 2.3). In  
3644 order to claim conformance with WS-HumanTask 1.1, the conformance targetes MUST comply with all  
3645 normative statements in this specification, notably all MUST statements have to be implemented.

3646

## A. Portability and Interoperability Considerations

3647

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

3648

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

3649

3650

- 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.

3651

3652

3653

3654

Portability requires support of WS-HumanTask artifacts.

3655

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

3656

3657

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

3658

3659

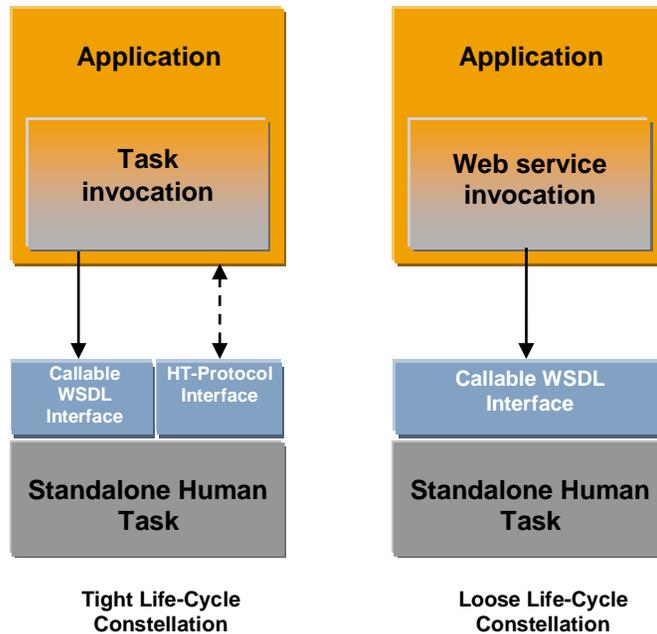
3660

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

3661

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

3662



3663

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

3664

3665

## B. WS-HumanTask Language Schema

```

3667 <?xml version="1.0" encoding="UTF-8"?>
3668 <!--
3669   Copyright (c) OASIS Open 2009. All Rights Reserved.
3670 -->
3671 <xsd:schema
3672   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3673   xmlns="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803"
3674   targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
3675   humantask/200803"
3676   elementFormDefault="qualified" blockDefault="#all">
3677
3678   <xsd:annotation>
3679     <xsd:documentation>
3680       XML Schema for WS-HumanTask 1.1 - WS-HumanTask Task Definition Language
3681     </xsd:documentation>
3682   </xsd:annotation>
3683
3684   <!-- other namespaces -->
3685   <xsd:import namespace="http://www.w3.org/XML/1998/namespace"
3686     schemaLocation="http://www.w3.org/2001/xml.xsd" />
3687
3688   <!-- base types for extensible elements -->
3689   <xsd:complexType name="tExtensibleElements">
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   <xsd:complexType name="tDocumentation" mixed="true">
3700     <xsd:sequence>
3701       <xsd:any namespace="##other" processContents="lax" minOccurs="0"
3702   maxOccurs="unbounded" />
3703     </xsd:sequence>
3704     <xsd:attribute ref="xml:lang" />
3705   </xsd:complexType>
3706
3707   <xsd:complexType name="tExtensibleMixedContentElements"
3708     mixed="true">
3709     <xsd:sequence>
3710       <xsd:element name="documentation" type="tDocumentation" minOccurs="0"
3711   maxOccurs="unbounded" />
3712       <xsd:any namespace="##other" processContents="lax" minOccurs="0"
3713   maxOccurs="unbounded" />
3714     </xsd:sequence>
3715     <xsd:anyAttribute namespace="##other" processContents="lax" />
3716   </xsd:complexType>
3717
3718   <!-- human interactions definition -->
3719   <xsd:element name="humanInteractions" type="tHumanInteractions" />
3720   <xsd:complexType name="tHumanInteractions">

```

```

3721     <xsd:complexContent>
3722         <xsd:extension base="tExtensibleElements">
3723             <xsd:sequence>
3724                 <xsd:element name="extensions" type="tExtensions" minOccurs="0" />
3725                 <xsd:element name="import" type="tImport" minOccurs="0"
3726 maxOccurs="unbounded" />
3727                 <xsd:element name="logicalPeopleGroups" type="tLogicalPeopleGroups"
3728 minOccurs="0" />
3729                 <xsd:element name="tasks" type="tTasks" minOccurs="0" />
3730                 <xsd:element name="notifications" type="tNotifications"
3731 minOccurs="0" />
3732             </xsd:sequence>
3733             <xsd:attribute name="targetNamespace" type="xsd:anyURI"
3734 use="required" />
3735             <xsd:attribute name="queryLanguage" type="xsd:anyURI" />
3736             <xsd:attribute name="expressionLanguage" type="xsd:anyURI" />
3737         </xsd:extension>
3738     </xsd:complexContent>
3739 </xsd:complexType>
3740
3741 <xsd:complexType name="tExtensions">
3742     <xsd:complexContent>
3743         <xsd:extension base="tExtensibleElements">
3744             <xsd:sequence>
3745                 <xsd:element name="extension" type="tExtension"
3746 maxOccurs="unbounded" />
3747             </xsd:sequence>
3748         </xsd:extension>
3749     </xsd:complexContent>
3750 </xsd:complexType>
3751
3752 <xsd:complexType name="tExtension">
3753     <xsd:complexContent>
3754         <xsd:extension base="tExtensibleElements">
3755             <xsd:attribute name="namespace" type="xsd:anyURI" use="required" />
3756             <xsd:attribute name="mustUnderstand" type="tBoolean" use="required"
3757 />
3758         </xsd:extension>
3759     </xsd:complexContent>
3760 </xsd:complexType>
3761
3762 <xsd:element name="import" type="tImport" />
3763 <xsd:complexType name="tImport">
3764     <xsd:complexContent>
3765         <xsd:extension base="tExtensibleElements">
3766             <xsd:attribute name="namespace" type="xsd:anyURI" use="optional" />
3767             <xsd:attribute name="location" type="xsd:anyURI" use="optional" />
3768             <xsd:attribute name="importType" type="xsd:anyURI" use="required" />
3769         </xsd:extension>
3770     </xsd:complexContent>
3771 </xsd:complexType>
3772
3773 <xsd:element name="logicalPeopleGroups" type="tLogicalPeopleGroups" />
3774 <xsd:complexType name="tLogicalPeopleGroups">
3775     <xsd:complexContent>
3776         <xsd:extension base="tExtensibleElements">
3777             <xsd:sequence>

```

```

3778         <xsd:element name="logicalPeopleGroup" type="tLogicalPeopleGroup"
3779 maxOccurs="unbounded" />
3780     </xsd:sequence>
3781 </xsd:extension>
3782 </xsd:complexContent>
3783 </xsd:complexType>
3784
3785 <xsd:complexType name="tLogicalPeopleGroup">
3786 <xsd:complexContent>
3787 <xsd:extension base="tExtensibleElements">
3788 <xsd:sequence>
3789 <xsd:element name="parameter" type="tParameter" minOccurs="0"
3790 maxOccurs="unbounded" />
3791 </xsd:sequence>
3792 <xsd:attribute name="name" type="xsd:NCName" use="required" />
3793 <xsd:attribute name="reference" type="xsd:NCName" use="optional" />
3794 </xsd:extension>
3795 </xsd:complexContent>
3796 </xsd:complexType>
3797
3798 <!-- generic human roles used in tasks and notifications -->
3799 <xsd:element name="genericHumanRole" type="tGenericHumanRoleAssignmentBase"
3800 abstract="true" block=""/>
3801
3802 <xsd:element name="potentialOwners" type="tPotentialOwnerAssignment"
3803 substitutionGroup="genericHumanRole"/>
3804 <xsd:element name="excludedOwners" type="tGenericHumanRoleAssignment"
3805 substitutionGroup="genericHumanRole"/>
3806 <xsd:element name="taskInitiator" type="tGenericHumanRoleAssignment"
3807 substitutionGroup="genericHumanRole"/>
3808 <xsd:element name="taskStakeholders" type="tGenericHumanRoleAssignment"
3809 substitutionGroup="genericHumanRole"/>
3810 <xsd:element name="businessAdministrators"
3811 type="tGenericHumanRoleAssignment" substitutionGroup="genericHumanRole"/>
3812 <xsd:element name="recipients" type="tGenericHumanRoleAssignment"
3813 substitutionGroup="genericHumanRole"/>
3814
3815 <xsd:complexType name="tGenericHumanRoleAssignmentBase" block="">
3816 <xsd:complexContent>
3817 <xsd:extension base="tExtensibleElements"/>
3818 </xsd:complexContent>
3819 </xsd:complexType>
3820
3821 <xsd:complexType name="tGenericHumanRoleAssignment">
3822 <xsd:complexContent>
3823 <xsd:extension base="tGenericHumanRoleAssignmentBase">
3824 <xsd:sequence>
3825 <xsd:element name="from" type="tFrom" />
3826 </xsd:sequence>
3827 </xsd:extension>
3828 </xsd:complexContent>
3829 </xsd:complexType>
3830
3831 <xsd:complexType name="tPotentialOwnerAssignment">
3832 <xsd:complexContent>
3833 <xsd:extension base="tGenericHumanRoleAssignmentBase">
3834 <xsd:choice>
3835 <xsd:element name="from" type="tFrom" />

```

```

3836     <xsd:element name="parallel" type="tParallel" />
3837     <xsd:element name="sequence" type="tSequence" />
3838   </xsd:choice>
3839 </xsd:extension>
3840 </xsd:complexContent>
3841 </xsd:complexType>
3842
3843 <!-- routing patterns -->
3844 <xsd:complexType name="tParallel">
3845   <xsd:complexContent>
3846     <xsd:extension base="tExtensibleElements">
3847       <xsd:sequence>
3848         <xsd:element name="completionBehavior" type="tCompletionBehavior"
3849 minOccurs="0" />
3850         <xsd:element name="from" type="tFrom" minOccurs="0"
3851 maxOccurs="unbounded" />
3852         <xsd:choice minOccurs="0" maxOccurs="unbounded">
3853           <xsd:element name="parallel" type="tParallel" />
3854           <xsd:element name="sequence" type="tSequence" />
3855         </xsd:choice>
3856       </xsd:sequence>
3857       <xsd:attribute name="type" type="tRoutingPatternType" />
3858     </xsd:extension>
3859   </xsd:complexContent>
3860 </xsd:complexType>
3861
3862 <xsd:complexType name="tSequence">
3863   <xsd:complexContent>
3864     <xsd:extension base="tExtensibleElements">
3865       <xsd:sequence>
3866         <xsd:element name="completionBehavior" type="tCompletionBehavior"
3867 />
3868         <xsd:element name="from" type="tFrom" minOccurs="0"
3869 maxOccurs="unbounded" />
3870         <xsd:choice minOccurs="0" maxOccurs="unbounded">
3871           <xsd:element name="parallel" type="tParallel" />
3872           <xsd:element name="sequence" type="tSequence" />
3873         </xsd:choice>
3874       </xsd:sequence>
3875       <xsd:attribute name="type" type="tRoutingPatternType" />
3876     </xsd:extension>
3877   </xsd:complexContent>
3878 </xsd:complexType>
3879
3880 <xsd:simpleType name="tRoutingPatternType">
3881   <xsd:restriction base="xsd:string">
3882     <xsd:enumeration value="all" />
3883     <xsd:enumeration value="single" />
3884   </xsd:restriction>
3885 </xsd:simpleType>
3886
3887 <!-- completion behavior -->
3888 <xsd:complexType name="tCompletionBehavior">
3889   <xsd:complexContent>
3890     <xsd:extension base="tExtensibleElements">
3891       <xsd:sequence>
3892         <xsd:element name="completion" type="tCompletion" minOccurs="0"
3893 maxOccurs="unbounded" />

```

```

3894         <xsd:element name="defaultCompletion" type="tDefaultCompletion"
3895 minOccurs="0" />
3896     </xsd:sequence>
3897     <xsd:attribute name="completionAction" type="tPattern" use="optional"
3898 default="automatic" />
3899 </xsd:extension>
3900 </xsd:complexContent>
3901 </xsd:complexType>
3902
3903 <xsd:complexType name="tCompletion">
3904 <xsd:complexContent>
3905 <xsd:extension base="tExtensibleElements">
3906 <xsd:sequence>
3907 <xsd:element name="condition" type="tBoolean-expr" />
3908 <xsd:element name="result" type="tResult" minOccurs="0" />
3909 </xsd:sequence>
3910 </xsd:extension>
3911 </xsd:complexContent>
3912 </xsd:complexType>
3913
3914 <xsd:complexType name="tDefaultCompletion">
3915 <xsd:complexContent>
3916 <xsd:extension base="tExtensibleElements">
3917 <xsd:sequence>
3918 <xsd:element name="result" type="tResult" />
3919 </xsd:sequence>
3920 </xsd:extension>
3921 </xsd:complexContent>
3922 </xsd:complexType>
3923
3924 <!-- result construction -->
3925 <xsd:complexType name="tResult">
3926 <xsd:complexContent>
3927 <xsd:extension base="tExtensibleElements">
3928 <xsd:choice maxOccurs="unbounded">
3929 <xsd:element name="aggregate" type="tAggregate" />
3930 <xsd:element name="copy" type="tCopy" />
3931 </xsd:choice>
3932 </xsd:extension>
3933 </xsd:complexContent>
3934 </xsd:complexType>
3935
3936 <xsd:complexType name="tAggregate">
3937 <xsd:complexContent>
3938 <xsd:extension base="tExtensibleElements">
3939 <xsd:attribute name="part" type="xsd:NCName" use="optional" />
3940 <xsd:attribute name="location" type="xsd:string" use="optional" />
3941 <xsd:attribute name="condition" type="xsd:string" />
3942 <xsd:attribute name="function" type="xsd:string" use="required" />
3943 </xsd:extension>
3944 </xsd:complexContent>
3945 </xsd:complexType>
3946
3947 <xsd:complexType name="tCopy">
3948 <xsd:complexContent>
3949 <xsd:extension base="tExtensibleElements">
3950 <xsd:sequence>
3951 <xsd:element name="from" type="tExpression" />

```

```

3952     <xsd:element name="to" type="tQuery" />
3953   </xsd:sequence>
3954 </xsd:extension>
3955 </xsd:complexContent>
3956 </xsd:complexType>
3957
3958 <!-- human tasks -->
3959 <xsd:element name="tasks" type="tTasks" />
3960 <xsd:complexType name="tTasks">
3961   <xsd:complexContent>
3962     <xsd:extension base="tExtensibleElements">
3963       <xsd:sequence>
3964         <xsd:element name="task" type="tTask" maxOccurs="unbounded" />
3965       </xsd:sequence>
3966     </xsd:extension>
3967   </xsd:complexContent>
3968 </xsd:complexType>
3969
3970 <xsd:complexType name="tTaskBase" abstract="true">
3971   <xsd:complexContent>
3972     <xsd:extension base="tExtensibleElements">
3973       <xsd:sequence>
3974         <xsd:element name="interface" type="tTaskInterface" minOccurs="0"
3975 />
3976         <xsd:element name="messageSchema" type="tMessageSchema"
3977 minOccurs="0" />
3978         <xsd:element name="priority" type="tPriority-expr" minOccurs="0" />
3979         <xsd:element name="peopleAssignments" type="tPeopleAssignments"
3980 minOccurs="0" />
3981         <xsd:element name="completionBehavior" type="tCompletionBehavior"
3982 minOccurs="0" />
3983         <xsd:element name="delegation" type="tDelegation" minOccurs="0" />
3984         <xsd:element name="presentationElements"
3985 type="tPresentationElements" minOccurs="0" />
3986         <xsd:element name="outcome" type="tQuery" minOccurs="0" />
3987         <xsd:element name="searchBy" type="tExpression" minOccurs="0" />
3988         <xsd:element name="renderings" type="tRenderings" minOccurs="0" />
3989         <xsd:element name="deadlines" type="tDeadlines" minOccurs="0" />
3990         <xsd:element name="composition" type="tComposition" minOccurs="0"
3991 />
3992       </xsd:sequence>
3993       <xsd:attribute name="name" type="xsd:NCName" use="required" />
3994       <xsd:attribute name="actualOwnerRequired" type="tBoolean"
3995 use="optional" default="yes" />
3996     </xsd:extension>
3997   </xsd:complexContent>
3998 </xsd:complexType>
3999
4000 <xsd:element name="task" type="tTask" />
4001 <xsd:complexType name="tTask">
4002   <xsd:complexContent>
4003     <xsd:restriction base="tTaskBase">
4004       <xsd:sequence>
4005         <xsd:element name="documentation" type="tDocumentation"
4006 minOccurs="0" maxOccurs="unbounded" />
4007         <xsd:any namespace="##other" processContents="lax" minOccurs="0"
4008 maxOccurs="unbounded" />
4009         <xsd:element name="interface" type="tTaskInterface" />

```

```

4010     <xsd:element name="messageSchema" type="tMessageSchema"
4011 minOccurs="0" maxOccurs="0" />
4012     <xsd:element name="priority" type="tPriority-expr" minOccurs="0" />
4013     <xsd:element name="peopleAssignments" type="tPeopleAssignments"
4014 minOccurs="0" />
4015     <xsd:element name="completionBehavior" type="tCompletionBehavior"
4016 minOccurs="0" />
4017     <xsd:element name="delegation" type="tDelegation" minOccurs="0" />
4018     <xsd:element name="presentationElements"
4019 type="tPresentationElements" minOccurs="0" />
4020     <xsd:element name="outcome" type="tQuery" minOccurs="0" />
4021     <xsd:element name="searchBy" type="tExpression" minOccurs="0" />
4022     <xsd:element name="renderings" type="tRenderings" minOccurs="0" />
4023     <xsd:element name="deadlines" type="tDeadlines" minOccurs="0" />
4024     <xsd:element name="composition" type="tComposition" minOccurs="0"
4025 />
4026     </xsd:sequence>
4027     <xsd:attribute name="name" type="xsd:NCName" use="required" />
4028     <xsd:attribute name="actualOwnerRequired" type="tBoolean"
4029 use="optional" default="yes" />
4030     <xsd:anyAttribute namespace="##other" processContents="lax" />
4031     </xsd:restriction>
4032 </xsd:complexContent>
4033 </xsd:complexType>
4034
4035 <xsd:complexType name="tTaskInterface">
4036 <xsd:complexContent>
4037 <xsd:extension base="tExtensibleElements">
4038 <xsd:attribute name="portType" type="xsd:QName" use="required" />
4039 <xsd:attribute name="operation" type="xsd:NCName" use="required" />
4040 <xsd:attribute name="responsePortType" type="xsd:QName"
4041 use="optional" />
4042 <xsd:attribute name="responseOperation" type="xsd:NCName"
4043 use="optional" />
4044 </xsd:extension>
4045 </xsd:complexContent>
4046 </xsd:complexType>
4047
4048 <!-- presentation elements -->
4049 <xsd:complexType name="tPresentationElements">
4050 <xsd:complexContent>
4051 <xsd:extension base="tExtensibleElements">
4052 <xsd:sequence>
4053 <xsd:element name="name" type="tText" minOccurs="0"
4054 maxOccurs="unbounded" />
4055 <xsd:element name="presentationParameters"
4056 type="tPresentationParameters" minOccurs="0" />
4057 <xsd:element name="subject" type="tText" minOccurs="0"
4058 maxOccurs="unbounded" />
4059 <xsd:element name="description" type="tDescription" minOccurs="0"
4060 maxOccurs="unbounded" />
4061 </xsd:sequence>
4062 </xsd:extension>
4063 </xsd:complexContent>
4064 </xsd:complexType>
4065
4066 <xsd:complexType name="tPresentationParameters">
4067 <xsd:complexContent>

```

```

4068     <xsd:extension base="tExtensibleElements">
4069         <xsd:sequence>
4070             <xsd:element name="presentationParameter"
4071 type="tPresentationParameter" maxOccurs="unbounded" />
4072         </xsd:sequence>
4073         <xsd:attribute name="expressionLanguage" type="xsd:anyURI" />
4074     </xsd:extension>
4075 </xsd:complexContent>
4076 </xsd:complexType>
4077
4078 <xsd:complexType name="tPresentationParameter">
4079     <xsd:complexContent>
4080         <xsd:extension base="tParameter" />
4081     </xsd:complexContent>
4082 </xsd:complexType>
4083
4084 <!-- elements for rendering tasks -->
4085 <xsd:complexType name="tRenderings">
4086     <xsd:complexContent>
4087         <xsd:extension base="tExtensibleElements">
4088             <xsd:sequence>
4089                 <xsd:element name="rendering" type="tRendering"
4090 maxOccurs="unbounded" />
4091             </xsd:sequence>
4092         </xsd:extension>
4093     </xsd:complexContent>
4094 </xsd:complexType>
4095
4096 <xsd:complexType name="tRendering">
4097     <xsd:complexContent>
4098         <xsd:extension base="tExtensibleElements">
4099             <xsd:attribute name="type" type="xsd:QName" use="required" />
4100         </xsd:extension>
4101     </xsd:complexContent>
4102 </xsd:complexType>
4103
4104 <!-- elements for people assignment -->
4105 <xsd:element name="peopleAssignments" type="tPeopleAssignments" />
4106 <xsd:complexType name="tPeopleAssignments">
4107     <xsd:complexContent>
4108         <xsd:extension base="tExtensibleElements">
4109             <xsd:sequence>
4110                 <xsd:element ref="genericHumanRole" minOccurs="0"
4111 maxOccurs="unbounded" />
4112             </xsd:sequence>
4113         </xsd:extension>
4114     </xsd:complexContent>
4115 </xsd:complexType>
4116
4117 <!-- elements for handling timeouts and escalation -->
4118 <xsd:complexType name="tDeadlines">
4119     <xsd:complexContent>
4120         <xsd:extension base="tExtensibleElements">
4121             <xsd:sequence>
4122                 <xsd:element name="startDeadline" type="tDeadline" minOccurs="0"
4123 maxOccurs="unbounded" />
4124                 <xsd:element name="completionDeadline" type="tDeadline"
4125 minOccurs="0" maxOccurs="unbounded" />

```

```

4126     </xsd:sequence>
4127     </xsd:extension>
4128     </xsd:complexContent>
4129 </xsd:complexType>
4130
4131 <xsd:complexType name="tDeadline">
4132     <xsd:complexContent>
4133         <xsd:extension base="tExtensibleElements">
4134             <xsd:sequence>
4135                 <xsd:choice>
4136                     <xsd:element name="for" type="tDuration-expr" />
4137                     <xsd:element name="until" type="tDeadline-expr" />
4138                 </xsd:choice>
4139                 <xsd:element name="escalation" type="tEscalation" minOccurs="0"
4140 maxOccurs="unbounded" />
4141             </xsd:sequence>
4142             <xsd:attribute name="name" type="xsd:NCName" use="required"/>
4143         </xsd:extension>
4144     </xsd:complexContent>
4145 </xsd:complexType>
4146
4147 <xsd:complexType name="tEscalation">
4148     <xsd:complexContent>
4149         <xsd:extension base="tExtensibleElements">
4150             <xsd:sequence>
4151                 <xsd:element name="condition" type="tBoolean-expr" minOccurs="0" />
4152                 <xsd:element name="toParts" type="tToParts" minOccurs="0" />
4153             <xsd:choice>
4154                 <xsd:element name="notification" type="tNotification" />
4155                 <xsd:element name="localNotification" type="tLocalNotification"
4156 />
4157                 <xsd:element name="reassignment" type="tReassignment" />
4158             </xsd:choice>
4159         </xsd:sequence>
4160         <xsd:attribute name="name" type="xsd:NCName" use="required" />
4161     </xsd:extension>
4162 </xsd:complexContent>
4163 </xsd:complexType>
4164
4165 <xsd:complexType name="tLocalNotification">
4166     <xsd:complexContent>
4167         <xsd:extension base="tExtensibleElements">
4168             <xsd:choice>
4169                 <xsd:sequence>
4170                     <xsd:element name="priority" type="tPriority-expr" minOccurs="0"
4171 />
4172                     <xsd:element name="peopleAssignments" type="tPeopleAssignments"
4173 minOccurs="0" />
4174                 </xsd:sequence>
4175             </xsd:choice>
4176             <xsd:attribute name="reference" type="xsd:QName" use="required" />
4177         </xsd:extension>
4178     </xsd:complexContent>
4179 </xsd:complexType>
4180
4181 <xsd:complexType name="tReassignment">
4182     <xsd:complexContent>
4183         <xsd:extension base="tExtensibleElements">

```

```

4184     <xsd:sequence>
4185         <xsd:element ref="potentialOwners" />
4186     </xsd:sequence>
4187 </xsd:extension>
4188 </xsd:complexContent>
4189 </xsd:complexType>
4190
4191 <xsd:complexType name="tToParts">
4192     <xsd:complexContent>
4193         <xsd:extension base="tExtensibleElements">
4194             <xsd:sequence>
4195                 <xsd:element name="toPart" type="tToPart" maxOccurs="unbounded" />
4196             </xsd:sequence>
4197         </xsd:extension>
4198     </xsd:complexContent>
4199 </xsd:complexType>
4200
4201 <xsd:complexType name="tToPart" mixed="true">
4202     <xsd:complexContent>
4203         <xsd:extension base="tExtensibleMixedContentElements">
4204             <xsd:attribute name="name" type="xsd:NCName" use="required" />
4205             <xsd:attribute name="expressionLanguage" type="xsd:anyURI" />
4206         </xsd:extension>
4207     </xsd:complexContent>
4208 </xsd:complexType>
4209
4210 <!-- task delegation -->
4211 <xsd:complexType name="tDelegation">
4212     <xsd:complexContent>
4213         <xsd:extension base="tExtensibleElements">
4214             <xsd:sequence>
4215                 <xsd:element name="from" type="tFrom" minOccurs="0" />
4216             </xsd:sequence>
4217             <xsd:attribute name="potentialDelegates" type="tPotentialDelegates"
4218 use="required" />
4219         </xsd:extension>
4220     </xsd:complexContent>
4221 </xsd:complexType>
4222
4223 <xsd:simpleType name="tPotentialDelegates">
4224     <xsd:restriction base="xsd:string">
4225         <xsd:enumeration value="anybody" />
4226         <xsd:enumeration value="nobody" />
4227         <xsd:enumeration value="potentialOwners" />
4228         <xsd:enumeration value="other" />
4229     </xsd:restriction>
4230 </xsd:simpleType>
4231
4232 <!-- composite tasks -->
4233 <xsd:complexType name="tComposition">
4234     <xsd:complexContent>
4235         <xsd:extension base="tExtensibleElements">
4236             <xsd:sequence>
4237                 <xsd:element name="subtask" type="tSubtask" maxOccurs="unbounded"
4238 />
4239             </xsd:sequence>
4240             <xsd:attribute name="type" type="tCompositionType" use="optional"
4241 default="sequential" />

```

```

4242     <xsd:attribute name="instantiationPattern" type="tPattern"
4243 use="optional" default="manual" />
4244     </xsd:extension>
4245   </xsd:complexContent>
4246 </xsd:complexType>
4247
4248 <xsd:simpleType name="tCompositionType">
4249   <xsd:restriction base="xsd:string">
4250     <xsd:enumeration value="sequential" />
4251     <xsd:enumeration value="parallel" />
4252   </xsd:restriction>
4253 </xsd:simpleType>
4254
4255 <xsd:simpleType name="tPattern">
4256   <xsd:restriction base="xsd:string">
4257     <xsd:enumeration value="manual" />
4258     <xsd:enumeration value="automatic" />
4259   </xsd:restriction>
4260 </xsd:simpleType>
4261
4262 <xsd:complexType name="tSubtask">
4263   <xsd:complexContent>
4264     <xsd:extension base="tExtensibleElements">
4265       <xsd:choice>
4266         <xsd:element name="task" type="tTask"/>
4267         <xsd:element name="localTask" type="tLocalTask" />
4268       </xsd:choice>
4269       <xsd:attribute name="name" type="xsd:NCName" use="required" />
4270     </xsd:extension>
4271   </xsd:complexContent>
4272 </xsd:complexType>
4273
4274 <xsd:complexType name="tLocalTask">
4275   <xsd:complexContent>
4276     <xsd:extension base="tExtensibleElements">
4277       <xsd:sequence>
4278         <xsd:element name="priority" type="tPriority-expr" minOccurs="0" />
4279         <xsd:element name="peopleAssignments" type="tPeopleAssignments"
4280 minOccurs="0" />
4281       </xsd:sequence>
4282       <xsd:attribute name="reference" type="xsd:QName" use="required" />
4283     </xsd:extension>
4284   </xsd:complexContent>
4285 </xsd:complexType>
4286
4287 <!-- lean tasks -->
4288 <xsd:element name="leanTask" type="tLeanTask"/>
4289 <xsd:complexType name="tLeanTask">
4290   <xsd:complexContent>
4291     <xsd:restriction base="tTaskBase">
4292       <xsd:sequence>
4293         <xsd:element name="documentation" type="tDocumentation"
4294 minOccurs="0" maxOccurs="unbounded" />
4295         <xsd:any namespace="##other" processContents="lax" minOccurs="0"
4296 maxOccurs="unbounded" />
4297         <xsd:element name="interface" type="tTaskInterface" minOccurs="0"
4298 maxOccurs="0" />
4299         <xsd:element name="messageSchema" type="tMessageSchema" />

```

```

4300     <xsd:element name="priority" type="tPriority-expr" minOccurs="0" />
4301     <xsd:element name="peopleAssignments" type="tPeopleAssignments"
4302 minOccurs="0" />
4303     <xsd:element name="delegation" type="tDelegation" minOccurs="0" />
4304     <xsd:element name="presentationElements"
4305 type="tPresentationElements" minOccurs="0" />
4306     <xsd:element name="outcome" type="tQuery" minOccurs="0" />
4307     <xsd:element name="searchBy" type="tExpression" minOccurs="0" />
4308     <xsd:element name="renderings" type="tRenderings" minOccurs="0" />
4309     <xsd:element name="deadlines" type="tDeadlines" minOccurs="0" />
4310     <xsd:element name="composition" type="tComposition" minOccurs="0"
4311 maxOccurs="0" />
4312   </xsd:sequence>
4313   <xsd:attribute name="name" type="xsd:NCName" use="required" />
4314   <xsd:attribute name="actualOwnerRequired" type="tBoolean"
4315 use="optional" default="yes" />
4316   <xsd:anyAttribute namespace="##other" processContents="lax" />
4317 </xsd:restriction>
4318 </xsd:complexContent>
4319 </xsd:complexType>
4320
4321 <xsd:complexType name="tMessageSchema">
4322   <xsd:complexContent>
4323     <xsd:extension base="tExtensibleElements">
4324       <xsd:sequence>
4325         <xsd:element name="messageField" type="tMessageField"
4326 minOccurs="0" maxOccurs="unbounded" />
4327       </xsd:sequence>
4328     </xsd:extension>
4329   </xsd:complexContent>
4330 </xsd:complexType>
4331
4332 <xsd:complexType name="tMessageField">
4333   <xsd:complexContent>
4334     <xsd:extension base="tExtensibleElements">
4335       <xsd:sequence>
4336         <xsd:element name="messageDisplay" type="tMessageDisplay"
4337 maxOccurs="unbounded" />
4338         <xsd:element name="messageChoice" type="tMessageChoice"
4339 minOccurs="0" maxOccurs="unbounded" />
4340       </xsd:sequence>
4341       <xsd:attribute name="name" type="xsd:NCName" />
4342       <xsd:attribute name="type" type="xsd:QName" />
4343     </xsd:extension>
4344   </xsd:complexContent>
4345 </xsd:complexType>
4346
4347 <xsd:complexType name="tMessageChoice">
4348   <xsd:complexContent>
4349     <xsd:extension base="tExtensibleElements">
4350       <xsd:sequence>
4351         <xsd:element name="messageDisplay" type="tMessageDisplay"
4352 maxOccurs="unbounded" />
4353       </xsd:sequence>
4354       <xsd:attribute name="value" type="xsd:anySimpleType" />
4355     </xsd:extension>
4356   </xsd:complexContent>
4357 </xsd:complexType>

```

```

4358
4359 | <xsd:complexType name="tMessageDisplay" mixed="true">
4360 |   <xsd:complexContent>
4361 |     <xsd:extension
4362 | base="tExtensibleElements<ExtensibleMixedContentElements">
4363 |       <xsd:attribute ref="xml:lang" />
4364 |     </xsd:extension>
4365 |   </xsd:complexContent>
4366 </xsd:complexType>
4367
4368 <!-- notifications -->
4369 <xsd:element name="notifications" type="tNotifications" />
4370 <xsd:complexType name="tNotifications">
4371 |   <xsd:complexContent>
4372 |     <xsd:extension base="tExtensibleElements">
4373 |       <xsd:sequence>
4374 |         <xsd:element name="notification" type="tNotification"
4375 | maxOccurs="unbounded" />
4376 |       </xsd:sequence>
4377 |     </xsd:extension>
4378 |   </xsd:complexContent>
4379 </xsd:complexType>
4380
4381 <xsd:element name="notification" type="tNotification" />
4382 <xsd:complexType name="tNotification">
4383 |   <xsd:complexContent>
4384 |     <xsd:extension base="tExtensibleElements">
4385 |       <xsd:sequence>
4386 |         <xsd:element name="interface" type="tNotificationInterface" />
4387 |         <xsd:element name="priority" type="tPriority-expr" minOccurs="0" />
4388 |         <xsd:element name="peopleAssignments" type="tPeopleAssignments" />
4389 |         <xsd:element name="presentationElements"
4390 | type="tPresentationElements" />
4391 |         <xsd:element name="renderings" type="tRenderings" minOccurs="0" />
4392 |       </xsd:sequence>
4393 |       <xsd:attribute name="name" type="xsd:NCName" use="required" />
4394 |     </xsd:extension>
4395 |   </xsd:complexContent>
4396 </xsd:complexType>
4397
4398 <xsd:complexType name="tNotificationInterface">
4399 |   <xsd:complexContent>
4400 |     <xsd:extension base="tExtensibleElements">
4401 |       <xsd:attribute name="portType" type="xsd:QName" use="required" />
4402 |       <xsd:attribute name="operation" type="xsd:NCName" use="required" />
4403 |     </xsd:extension>
4404 |   </xsd:complexContent>
4405 </xsd:complexType>
4406
4407 <!-- miscellaneous helper types -->
4408 <xsd:complexType name="tText" mixed="true">
4409 |   <xsd:complexContent>
4410 |     <xsd:extension base="tExtensibleMixedContentElements">
4411 |       <xsd:attribute ref="xml:lang" />
4412 |     </xsd:extension>
4413 |   </xsd:complexContent>
4414 </xsd:complexType>
4415

```

```

4416 <xsd:complexType name="tDescription" mixed="true">
4417   <xsd:complexContent>
4418     <xsd:extension base="tExtensibleMixedContentElements">
4419       <xsd:attribute ref="xml:lang" />
4420       <xsd:attribute name="contentType" type="xsd:string" />
4421     </xsd:extension>
4422   </xsd:complexContent>
4423 </xsd:complexType>
4424
4425 <xsd:complexType name="tFrom" mixed="true">
4426   <xsd:complexContent>
4427     <xsd:extension base="tExtensibleMixedContentElements">
4428       <xsd:sequence>
4429         <xsd:choice>
4430           <xsd:element name="argument" type="tArgument" minOccurs="0"
4431 maxOccurs="unbounded"/>
4432           <xsd:element name="literal" type="tLiteral" minOccurs="0" />
4433         </xsd:choice>
4434       </xsd:sequence>
4435       <xsd:attribute name="expressionLanguage" type="xsd:anyURI" />
4436       <xsd:attribute name="logicalPeopleGroup" type="xsd:NCName" />
4437     </xsd:extension>
4438   </xsd:complexContent>
4439 </xsd:complexType>
4440
4441 <xsd:complexType name="tArgument">
4442   <xsd:complexContent>
4443     <xsd:extension base="tExtensibleMixedContentElements">
4444       <xsd:attribute name="name" type="xsd:NCName" />
4445       <xsd:attribute name="expressionLanguage" type="xsd:anyURI" />
4446     </xsd:extension>
4447   </xsd:complexContent>
4448 </xsd:complexType>
4449
4450 <xsd:complexType name="tParameter" mixed="true">
4451   <xsd:complexContent>
4452     <xsd:extension base="tExtensibleMixedContentElements">
4453       <xsd:attribute name="name" type="xsd:NCName" use="required" />
4454       <xsd:attribute name="type" type="xsd:QName" use="required" />
4455     </xsd:extension>
4456   </xsd:complexContent>
4457 </xsd:complexType>
4458
4459 <xsd:complexType name="tLiteral" mixed="true">
4460   <xsd:sequence>
4461     <xsd:any namespace="##any" processContents="lax"/>
4462   </xsd:sequence>
4463   <xsd:anyAttribute namespace="##other" processContents="lax" />
4464 </xsd:complexType>
4465
4466 <xsd:complexType name="tQuery" mixed="true">
4467   <xsd:complexContent>
4468     <xsd:extension base="tExtensibleMixedContentElements">
4469       <xsd:attribute name="part" />
4470       <xsd:attribute name="queryLanguage" type="xsd:anyURI" />
4471     </xsd:extension>
4472   </xsd:complexContent>
4473 </xsd:complexType>

```

```

4474 <xsd:complexType name="tExpression" mixed="true">
4475   <xsd:complexContent>
4476     <xsd:extension base="tExtensibleMixedContentElements">
4477       <xsd:attribute name="expressionLanguage" type="xsd:anyURI" />
4478     </xsd:extension>
4479   </xsd:complexContent>
4480 </xsd:complexType>
4481
4482 <xsd:element name="priority" type="tPriority-expr" />
4483 <xsd:complexType name="tPriority-expr" mixed="true">
4484   <xsd:complexContent mixed="true">
4485     <xsd:extension base="tExpression" />
4486   </xsd:complexContent>
4487 </xsd:complexType>
4488
4489 <xsd:complexType name="tBoolean-expr" mixed="true">
4490   <xsd:complexContent mixed="true">
4491     <xsd:extension base="tExpression" />
4492   </xsd:complexContent>
4493 </xsd:complexType>
4494
4495 <xsd:complexType name="tDuration-expr" mixed="true">
4496   <xsd:complexContent mixed="true">
4497     <xsd:extension base="tExpression" />
4498   </xsd:complexContent>
4499 </xsd:complexType>
4500
4501 <xsd:complexType name="tDeadline-expr" mixed="true">
4502   <xsd:complexContent mixed="true">
4503     <xsd:extension base="tExpression" />
4504   </xsd:complexContent>
4505 </xsd:complexType>
4506
4507 <xsd:simpleType name="tBoolean">
4508   <xsd:restriction base="xsd:string">
4509     <xsd:enumeration value="yes" />
4510     <xsd:enumeration value="no" />
4511   </xsd:restriction>
4512 </xsd:simpleType>
4513
4514 </xsd:schema>
4515

```

4516

## C. WS-HumanTask Data Types Schema

```
4517 <?xml version="1.0" encoding="UTF-8"?>
4518 <!--
4519 Copyright (c) OASIS Open 2009. All Rights Reserved.
4520 -->
4521 <xsd:schema
4522   targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
4523 humantask/types/200803"
4524   xmlns="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/types/200803"
4525   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
4526   elementFormDefault="qualified"
4527   blockDefault="#all">
4528
4529   <xsd:annotation>
4530     <xsd:documentation>
4531       XML Schema for WS-HumanTask 1.1 - WS-HumanTask Data Type Definitions
4532     </xsd:documentation>
4533   </xsd:annotation>
4534
4535   <!-- other namespaces -->
4536   <xsd:import namespace="http://www.w3.org/XML/1998/namespace"
4537 schemaLocation="http://www.w3.org/2001/xml.xsd"/>
4538
4539   <!-- data types for attachment operations -->
4540   <xsd:element name="attachmentInfo" type="tAttachmentInfo"/>
4541   <xsd:complexType name="tAttachmentInfo">
4542     <xsd:sequence>
4543       <xsd:element name="identifier" type="xsd:anyURI"/>
4544       <xsd:element name="name" type="xsd:string"/>
4545       <xsd:element name="accessType" type="xsd:string"/>
4546       <xsd:element name="contentType" type="xsd:string"/>
4547       <xsd:element name="contentCategory" type="xsd:anyURI"/>
4548       <xsd:element name="attachedTime" type="xsd:dateTime"/>
4549       <xsd:element name="attachedBy" type="tUser"/>
4550       <xsd:any namespace="##other" processContents="lax" minOccurs="0"
4551 maxOccurs="unbounded"/>
4552     </xsd:sequence>
4553   </xsd:complexType>
4554   <xsd:element name="attachment" type="tAttachment"/>
4555   <xsd:complexType name="tAttachment">
4556     <xsd:sequence>
4557       <xsd:element ref="attachmentInfo"/>
4558       <xsd:element name="value" type="xsd:anyType"/>
4559     </xsd:sequence>
4560   </xsd:complexType>
4561
4562   <!-- data types for comments -->
4563   <xsd:element name="comment" type="tComment"/>
4564   <xsd:complexType name="tComment">
4565     <xsd:sequence>
4566       <xsd:element name="id" type="xsd:anyURI"/>
4567       <xsd:element name="addedTime" type="xsd:dateTime"/>
4568       <xsd:element name="addedBy" type="tUser"/>
4569       <xsd:element name="lastModifiedTime" type="xsd:dateTime"/>
4570     </xsd:sequence>
4571   </xsd:complexType>
4572 </xsd:schema>
```

```

4570     <xsd:element name="lastModifiedBy" type="tUser"/>
4571     <xsd:element name="text" type="xsd:string"/>
4572     <xsd:any namespace="##other" processContents="lax" minOccurs="0"
4573 maxOccurs="unbounded"/>
4574   </xsd:sequence>
4575 </xsd:complexType>
4576
4577 <!-- data types for simple query operations -->
4578 <xsd:element name="taskAbstract" type="tTaskAbstract"/>
4579 <xsd:complexType name="tTaskAbstract">
4580   <xsd:sequence>
4581     <xsd:element name="id" type="xsd:anyURI"/>
4582     <xsd:element name="taskType" type="xsd:string"/>
4583     <xsd:element name="name" type="xsd:QName"/>
4584     <xsd:element name="status" type="tStatus"/>
4585     <xsd:element name="priority" type="tPriority" minOccurs="0"/>
4586     <xsd:element name="createdTime" type="xsd:dateTime"/>
4587     <xsd:element name="activationTime" type="xsd:dateTime" minOccurs="0"/>
4588     <xsd:element name="expirationTime" type="xsd:dateTime" minOccurs="0"/>
4589     <xsd:element name="isSkipable" type="xsd:boolean" minOccurs="0"/>
4590     <xsd:element name="hasPotentialOwners" type="xsd:boolean"
4591 minOccurs="0"/>
4592     <xsd:element name="startByTimeExists" type="xsd:boolean"
4593 minOccurs="0"/>
4594     <xsd:element name="completeByTimeExists" type="xsd:boolean"
4595 minOccurs="0"/>
4596     <xsd:element name="presentationName" type="tPresentationName"
4597 minOccurs="0"/>
4598     <xsd:element name="presentationSubject" type="tPresentationSubject"
4599 minOccurs="0"/>
4600     <xsd:element name="renderingMethodExists" type="xsd:boolean"/>
4601     <xsd:element name="hasOutput" type="xsd:boolean" minOccurs="0"/>
4602     <xsd:element name="hasFault" type="xsd:boolean" minOccurs="0"/>
4603     <xsd:element name="hasAttachments" type="xsd:boolean" minOccurs="0"/>
4604     <xsd:element name="hasComments" type="xsd:boolean" minOccurs="0"/>
4605     <xsd:element name="escalated" type="xsd:boolean" minOccurs="0"/>
4606     <xsd:element name="outcome" type="xsd:string" minOccurs="0"/>
4607     <xsd:element name="parentTaskId" type="xsd:anyURI" minOccurs="0"/>
4608     <xsd:element name="hasSubTasks" type="xsd:boolean" minOccurs="0"/>
4609     <xsd:any namespace="##other" processContents="lax" minOccurs="0"
4610 maxOccurs="unbounded"/>
4611   </xsd:sequence>
4612 </xsd:complexType>
4613 <xsd:element name="taskDetails" type="tTaskDetails"/>
4614 <xsd:complexType name="tTaskDetails">
4615   <xsd:sequence>
4616     <xsd:element name="id" type="xsd:anyURI"/>
4617     <xsd:element name="taskType" type="xsd:string"/>
4618     <xsd:element name="name" type="xsd:QName"/>
4619     <xsd:element name="status" type="tStatus"/>
4620     <xsd:element name="priority" type="tPriority" minOccurs="0"/>
4621     <xsd:element name="taskInitiator" type="tUser" minOccurs="0"/>
4622     <xsd:element name="taskStakeholders" type="tOrganizationalEntity"
4623 minOccurs="0"/>
4624     <xsd:element name="potentialOwners" type="tOrganizationalEntity"
4625 minOccurs="0"/>
4626     <xsd:element name="businessAdministrators" type="tOrganizationalEntity"
4627 minOccurs="0"/>

```

```

4628     <xsd:element name="actualOwner" type="tUser" minOccurs="0"/>
4629     <xsd:element name="notificationRecipients" type="tOrganizationalEntity"
4630 minOccurs="0"/>
4631     <xsd:element name="createdTime" type="xsd:dateTime"/>
4632     <xsd:element name="createdBy" type="xsd:stringtUser" minOccurs="0"/>
4633     <xsd:element name="lastModifiedTime" type="xsd:dateTime"/>
4634     <xsd:element name="lastModifiedBy" type="xsd:stringtUser"
4635 minOccurs="0"/>
4636     <xsd:element name="activationTime" type="xsd:dateTime" minOccurs="0"/>
4637     <xsd:element name="expirationTime" type="xsd:dateTime" minOccurs="0"/>
4638     <xsd:element name="isSkipable" type="xsd:boolean" minOccurs="0"/>
4639     <xsd:element name="hasPotentialOwners" type="xsd:boolean"
4640 minOccurs="0"/>
4641     <xsd:element name="startByTimeExists" type="xsd:boolean"
4642 minOccurs="0"/>
4643     <xsd:element name="completeByTimeExists" type="xsd:boolean"
4644 minOccurs="0"/>
4645     <xsd:element name="presentationName" type="tPresentationName"
4646 minOccurs="0"/>
4647     <xsd:element name="presentationSubject" type="tPresentationSubject"
4648 minOccurs="0"/>
4649     <xsd:element name="renderingMethodExists" type="xsd:boolean"/>
4650     <xsd:element name="hasOutput" type="xsd:boolean" minOccurs="0"/>
4651     <xsd:element name="hasFault" type="xsd:boolean" minOccurs="0"/>
4652     <xsd:element name="hasAttachments" type="xsd:boolean" minOccurs="0"/>
4653     <xsd:element name="hasComments" type="xsd:boolean" minOccurs="0"/>
4654     <xsd:element name="escalated" type="xsd:boolean" minOccurs="0"/>
4655     <xsd:element name="searchBy" type="xsd:string" minOccurs="0"/>
4656     <xsd:element name="outcome" type="xsd:string" minOccurs="0"/>
4657     <xsd:element name="parentTaskId" type="xsd:anyURI" minOccurs="0"/>
4658     <xsd:element name="hasSubTasks" type="xsd:boolean" minOccurs="0"/>
4659     <xsd:any namespace="##other" processContents="lax" minOccurs="0"
4660 maxOccurs="unbounded"/>
4661 </xsd:sequence>
4662 </xsd:complexType>
4663 <xsd:simpleType name="tPresentationName">
4664   <xsd:annotation>
4665     <xsd:documentation>length-restricted string</xsd:documentation>
4666   </xsd:annotation>
4667   <xsd:restriction base="xsd:string">
4668     <xsd:maxLength value="64"/>
4669     <xsd:whiteSpace value="preserve"/>
4670   </xsd:restriction>
4671 </xsd:simpleType>
4672 <xsd:simpleType name="tPresentationSubject">
4673   <xsd:annotation>
4674     <xsd:documentation>length-restricted type string</xsd:documentation>
4675   </xsd:annotation>
4676   <xsd:restriction base="xsd:string">
4677     <xsd:maxLength value="254"/>
4678     <xsd:whiteSpace value="preserve"/>
4679   </xsd:restriction>
4680 </xsd:simpleType>
4681 <xsd:simpleType name="tStatus">
4682   <xsd:restriction base="xsd:string"/>
4683 </xsd:simpleType>
4684 <xsd:simpleType name="tPredefinedStatus">
4685   <xsd:annotation>

```

```

4686     <xsd:documentation>for documentation only</xsd:documentation>
4687 </xsd:annotation>
4688 <xsd:restriction base="xsd:string">
4689   <xsd:enumeration value="CREATED"/>
4690   <xsd:enumeration value="READY"/>
4691   <xsd:enumeration value="RESERVED"/>
4692   <xsd:enumeration value="IN_PROGRESS"/>
4693   <xsd:enumeration value="SUSPENDED"/>
4694   <xsd:enumeration value="COMPLETED"/>
4695   <xsd:enumeration value="FAILED"/>
4696   <xsd:enumeration value="ERROR"/>
4697   <xsd:enumeration value="EXITED"/>
4698   <xsd:enumeration value="OBSOLETE"/>
4699 </xsd:restriction>
4700 </xsd:simpleType>
4701 <xsd:simpleType name="tPriority">
4702   <xsd:restriction base="xsd:integer">
4703     <xsd:minInclusive value="0"/>
4704     <xsd:maxInclusive value="10"/>
4705   </xsd:restriction>
4706 </xsd:simpleType>
4707 <xsd:complexType name="tTime">
4708   <xsd:choice>
4709     <xsd:element name="timePeriod" type="xsd:duration"/>
4710     <xsd:element name="pointOfTime" type="xsd:dateTime"/>
4711   </xsd:choice>
4712 </xsd:complexType>
4713
4714 <!-- task operations -->
4715 <xsd:complexType name="tTaskOperations">
4716   <xsd:choice maxOccurs="unbounded">
4717     <xsd:element name="activate" type="tTaskOperation"/>
4718     <xsd:element name="addAttachment" type="tTaskOperation"/>
4719     <xsd:element name="addComment" type="tTaskOperation"/>
4720     <xsd:element name="claim" type="tTaskOperation"/>
4721     <xsd:element name="complete" type="tTaskOperation"/>
4722     <xsd:element name="delegate" type="tTaskOperation"/>
4723     <xsd:element name="deleteAttachment" type="tTaskOperation"/>
4724     <xsd:element name="deleteComment" type="tTaskOperation"/>
4725     <xsd:element name="deleteFault" type="tTaskOperation"/>
4726     <xsd:element name="deleteOutput" type="tTaskOperation"/>
4727     <xsd:element name="fail" type="tTaskOperation"/>
4728     <xsd:element name="forward" type="tTaskOperation"/>
4729     <xsd:element name="getAttachment" type="tTaskOperation"/>
4730     <xsd:element name="getAttachmentInfos" type="tTaskOperation"/>
4731     <xsd:element name="getComments" type="tTaskOperation"/>
4732     <xsd:element name="getFault" type="tTaskOperation"/>
4733     <xsd:element name="getInput" type="tTaskOperation"/>
4734     <xsd:element name="getOutcome" type="tTaskOperation"/>
4735     <xsd:element name="getOutput" type="tTaskOperation"/>
4736     <xsd:element name="getParentTask" type="tTaskOperation"/>
4737     <xsd:element name="getParentTaskIdentifier" type="tTaskOperation"/>
4738     <xsd:element name="getRendering" type="tTaskOperation"/>
4739     <xsd:element name="getRenderingTypes" type="tTaskOperation"/>
4740     <xsd:element name="getSubtaskIdentifiers" type="tTaskOperation"/>
4741     <xsd:element name="getSubtasks" type="tTaskOperation"/>
4742     <xsd:element name="getTaskDescription" type="tTaskOperation"/>
4743     <xsd:element name="getTaskDetails" type="tTaskOperation"/>

```

```

4744     <xsd:element name="getTaskHistory" type="tTaskOperation"/>
4745     <xsd:element name="getTaskInstanceData" type="tTaskOperation"/>
4746     <xsd:element name="hasSubtasks" type="tTaskOperation"/>
4747     <xsd:element name="instantiateSubtask" type="tTaskOperation"/>
4748     <xsd:element name="isSubtask" type="tTaskOperation"/>
4749     <xsd:element name="nominate" type="tTaskOperation"/>
4750     <xsd:element name="release" type="tTaskOperation"/>
4751     <xsd:element name="remove" type="tTaskOperation"/>
4752     <xsd:element name="resume" type="tTaskOperation"/>
4753     <xsd:element name="setFault" type="tTaskOperation"/>
4754     <xsd:element name="setGenericHumanRole" type="tTaskOperation"/>
4755     <xsd:element name="setOutput" type="tTaskOperation"/>
4756     <xsd:element name="setPriority" type="tTaskOperation"/>
4757     <xsd:element name="setTaskCompletionDeadlineExpression"
4758 type="tTaskOperation"/>
4759     <xsd:element name="setTaskCompletionDurationExpression"
4760 type="tTaskOperation"/>
4761     <xsd:element name="setTaskStartDeadlineExpression"
4762 type="tTaskOperation"/>
4763     <xsd:element name="setTaskStartDurationExpression"
4764 type="tTaskOperation"/>
4765     <xsd:element name="skip" type="tTaskOperation"/>
4766     <xsd:element name="start" type="tTaskOperation"/>
4767     <xsd:element name="stop" type="tTaskOperation"/>
4768     <xsd:element name="suspend" type="tTaskOperation"/>
4769     <xsd:element name="suspendUntil" type="tTaskOperation"/>
4770     <xsd:element name="updateComment" type="tTaskOperation"/>
4771     <xsd:any namespace="##other" processContents="lax"/>
4772 </xsd:choice>
4773 </xsd:complexType>
4774 <xsd:complexType name="tTaskOperation">
4775   <xsd:complexContent>
4776     <xsd:restriction base="xsd:anyType"/>
4777   </xsd:complexContent>
4778 </xsd:complexType>
4779
4780 <!-- data types for advanced query operations -->
4781 <xsd:element name="taskQueryResultSet" type="tTaskQueryResultSet"/>
4782 <xsd:complexType name="tTaskQueryResultSet">
4783   <xsd:sequence>
4784     <xsd:element name="row" type="tTaskQueryResultRow" minOccurs="0"
4785 maxOccurs="unbounded"/>
4786   </xsd:sequence>
4787 </xsd:complexType>
4788 <xsd:complexType name="tTaskQueryResultRow">
4789   <xsd:choice minOccurs="0" maxOccurs="unbounded">
4790     <xsd:element name="id" type="xsd:anyURI"/>
4791     <xsd:element name="taskType" type="xsd:string"/>
4792     <xsd:element name="name" type="xsd:QName"/>
4793     <xsd:element name="status" type="tStatus"/>
4794     <xsd:element name="priority" type="tPriority"/>
4795     <xsd:element name="taskInitiator" type="tOrganizationalEntity"/>
4796     <xsd:element name="taskStakeholders" type="tOrganizationalEntity"/>
4797     <xsd:element name="potentialOwners" type="tOrganizationalEntity"/>
4798     <xsd:element name="businessAdministrators"
4799 type="tOrganizationalEntity"/>
4800     <xsd:element name="actualOwner" type="tUser"/>

```

```

4801     <xsd:element name="notificationRecipients"
4802 type="tOrganizationalEntity"/>
4803     <xsd:element name="createdTime" type="xsd:dateTime"/>
4804     <xsd:element name="createdBy" type="xsd:stringtUser"/>
4805     <xsd:element name="lastModifiedTime" type="xsd:dateTime"/>
4806     <xsd:element name="lastModifiedBy" type="xsd:stringtUser"/>
4807     <xsd:element name="activationTime" type="xsd:dateTime"/>
4808     <xsd:element name="expirationTime" type="xsd:dateTime"/>
4809     <xsd:element name="isSkipable" type="xsd:boolean"/>
4810     <xsd:element name="hasPotentialOwners" type="xsd:boolean"/>
4811     <xsd:element name="startByTime" type="xsd:dateTime"/>
4812     <xsd:element name="completeByTime" type="xsd:dateTime"/>
4813     <xsd:element name="presentationName" type="tPresentationName"/>
4814     <xsd:element name="presentationSubject" type="tPresentationSubject"/>
4815     <xsd:element name="renderingMethodName" type="xsd:QName"/>
4816     <xsd:element name="hasOutput" type="xsd:boolean"/>
4817     <xsd:element name="hasFault" type="xsd:boolean"/>
4818     <xsd:element name="hasAttachments" type="xsd:boolean"/>
4819     <xsd:element name="hasComments" type="xsd:boolean"/>
4820     <xsd:element name="escalated" type="xsd:boolean"/>
4821     <xsd:element name="parentTaskId" type="xsd:anyURI"/>
4822     <xsd:element name="hasSubtasks" type="xsd:boolean"/>
4823     <xsd:element name="searchBy" type="xsd:string"/>
4824     <xsd:element name="outcome" type="xsd:string"/>
4825     <xsd:element name="taskOperations" type="tTaskOperations"/>
4826     <xsd:any namespace="##other" processContents="lax"/>
4827 </xsd:choice>
4828 </xsd:complexType>
4829 <xsd:complexType name="tFault">
4830   <xsd:sequence>
4831     <xsd:element name="faultName" type="xsd:NCName"/>
4832     <xsd:element name="faultData" type="xsd:anyType"/>
4833   </xsd:sequence>
4834 </xsd:complexType>
4835
4836 <!-- elements and types for organizational entities -->
4837 <xsd:element name="organizationalEntity" type="tOrganizationalEntity"/>
4838 <xsd:complexType name="tOrganizationalEntity">
4839   <xsd:choice maxOccurs="unbounded">
4840     <xsd:element name="user" type="tUser"/>
4841     <xsd:element name="group" type="tGroup"/>
4842   </xsd:choice>
4843 </xsd:complexType>
4844 <xsd:element name="user" type="tUser"/>
4845 <xsd:simpleType name="tUser">
4846   <xsd:restriction base="xsd:string"/>
4847 </xsd:simpleType>
4848 <xsd:element name="group" type="tGroup"/>
4849 <xsd:simpleType name="tGroup">
4850   <xsd:restriction base="xsd:string"/>
4851 </xsd:simpleType>
4852
4853 <!-- input or output message part data -->
4854 <xsd:element name="part" type="tPart"/>
4855 <xsd:complexType name="tPart" mixed="true">
4856   <xsd:sequence>
4857     <xsd:any processContents="skip" minOccurs="0"/>
4858   </xsd:sequence>

```

```

4859     <xsd:attribute name="name" type="xsd:NCName" use="required"/>
4860 </xsd:complexType>
4861
4862 <!-- type container element for one or more message parts -->
4863 <xsd:complexType name="tMessagePartsData">
4864     <xsd:sequence>
4865         <xsd:element ref="part" minOccurs="0" maxOccurs="unbounded"/>
4866     </xsd:sequence>
4867 </xsd:complexType>
4868 <xsd:complexType name="tFaultData">
4869     <xsd:sequence>
4870         <xsd:element name="faultName" type="xsd:NCName"/>
4871         <xsd:element name="faultData" type="xsd:anyType"/>
4872     </xsd:sequence>
4873 </xsd:complexType>
4874 <xsd:element name="attachmentInfos" type="tAttachmentInfos"/>
4875 <xsd:complexType name="tAttachmentInfos">
4876     <xsd:sequence>
4877         <xsd:element name="info" type="tAttachmentInfo" minOccurs="0"
4878 maxOccurs="unbounded"/>
4879     </xsd:sequence>
4880 </xsd:complexType>
4881 <xsd:element name="comments" type="tComments"/>
4882 <xsd:complexType name="tComments">
4883     <xsd:sequence>
4884         <xsd:element ref="comment" minOccurs="0" maxOccurs="unbounded"/>
4885     </xsd:sequence>
4886 </xsd:complexType>
4887 <xsd:element name="renderingType" type="xsd:QName"/>
4888 <xsd:complexType name="tRenderingTypes">
4889     <xsd:sequence>
4890         <xsd:element ref="renderingType" minOccurs="0" maxOccurs="unbounded"/>
4891     </xsd:sequence>
4892 </xsd:complexType>
4893
4894 <!-- Single rendering element that contains rendering type (attribute) and
4895 data. -->
4896 <xsd:element name="rendering" type="tRendering"/>
4897 <xsd:complexType name="tRendering">
4898     <xsd:sequence>
4899         <xsd:any namespace="##other" processContents="lax" minOccurs="0"
4900 maxOccurs="unbounded"/>
4901     </xsd:sequence>
4902     <xsd:attribute name="type" type="xsd:QName" use="required"/>
4903 </xsd:complexType>
4904 <xsd:element name="renderings">
4905     <xsd:complexType>
4906         <xsd:sequence>
4907             <xsd:element ref="rendering" minOccurs="0" maxOccurs="unbounded"/>
4908         </xsd:sequence>
4909     </xsd:complexType>
4910 </xsd:element>
4911 <xsd:element name="description" type="xsd:string"/>
4912 <xsd:complexType name="tTaskInstanceData">
4913     <xsd:sequence>
4914         <!-- taskDetails contains task ID, meta data, presentation name and
4915 presentation subject. -->
4916         <xsd:element ref="taskDetails"/>

```

```

4917     <xsd:element ref="description"/>
4918     <xsd:element name="input" type="tMessagePartsData"/>
4919     <xsd:element name="output" type="tMessagePartsData" nillable="true"/>
4920     <xsd:element name="fault" type="tFaultData" nillable="true"
4921 minOccurs="0"/>
4922     <xsd:element ref="renderings" minOccurs="0"/>
4923     <xsd:element ref="comments" minOccurs="0"/>
4924     <xsd:element ref="attachmentInfos" minOccurs="0"/>
4925     <xsd:any namespace="##other" processContents="lax" minOccurs="0"
4926 maxOccurs="unbounded"/>
4927   </xsd:sequence>
4928 </xsd:complexType>
4929
4930 <!-- Defines the human task event types -->
4931 <xsd:simpleType name="tTaskEventType">
4932   <xsd:restriction base="xsd:string">
4933     <xsd:enumeration value="create"/>
4934     <xsd:enumeration value="claim"/>
4935     <xsd:enumeration value="start"/>
4936     <xsd:enumeration value="stop"/>
4937     <xsd:enumeration value="release"/>
4938     <xsd:enumeration value="suspend"/>
4939     <xsd:enumeration value="suspendUntil"/>
4940     <xsd:enumeration value="resume"/>
4941     <xsd:enumeration value="complete"/>
4942     <xsd:enumeration value="remove"/>
4943     <xsd:enumeration value="fail"/>
4944     <xsd:enumeration value="setPriority"/>
4945     <xsd:enumeration value="addAttachment"/>
4946     <xsd:enumeration value="deleteattachment"/>
4947     <xsd:enumeration value="addComment"/>
4948     <xsd:enumeration value="skip"/>
4949     <xsd:enumeration value="forward"/>
4950     <xsd:enumeration value="delegate"/>
4951     <xsd:enumeration value="setOutput"/>
4952     <xsd:enumeration value="deleteOutput"/>
4953     <xsd:enumeration value="setFault"/>
4954     <xsd:enumeration value="deleteFault"/>
4955     <xsd:enumeration value="activate"/>
4956     <xsd:enumeration value="nominate"/>
4957     <xsd:enumeration value="setGenericHumanRole"/>
4958     <xsd:enumeration value="expire"/>
4959     <xsd:enumeration value="escalated"/>
4960   </xsd:restriction>
4961 </xsd:simpleType>
4962 <xsd:element name="taskEvent">
4963   <xsd:complexType>
4964     <xsd:annotation>
4965       <xsd:documentation>
4966         A detailed event that represents a change in the task's state.
4967       </xsd:documentation>
4968     </xsd:annotation>
4969     <xsd:sequence>
4970       <!-- event id - unique per task -->
4971       <xsd:element name="id" type="xsd:integer"/>
4972       <!-- event date time -->
4973       <xsd:element name="eventTime" type="xsd:dateTime"/>
4974       <!-- task ID -->

```

```

4975     <xsd:element name="identifier" type="xsd:anyURI"/>
4976     <xsd:element name="principal" type="xsd:string" nillable="true"
4977 minOccurs="0"/>
4978     <!-- Event type. Note - using a restricted type limits extensibility
4979 to add custom event types. -->
4980     <xsd:element name="eventType" type="tTaskEventType"/>
4981     <!-- actual owner of the task before the event -->
4982     <xsd:element name="startOwner" type="xsd:string" nillable="true"
4983 minOccurs="0"/>
4984     <!-- actual owner of the task after the event -->
4985     <xsd:element name="endOwner" type="xsd:string" nillable="true"
4986 minOccurs="0"/>
4987     <!-- WSH task status -->
4988     <xsd:element name="status" type="tStatus"/>
4989     <!-- boolean to indicate this event has optional data -->
4990     <xsd:element name="hasData" type="xsd:boolean" minOccurs="0"/>
4991     <xsd:element name="eventData" type="xsd:anyType" nillable="true"
4992 minOccurs="0"/>
4993     <xsd:element name="faultName" type="xsd:string" nillable="true"
4994 minOccurs="0"/>
4995     <!-- extensibility -->
4996     <xsd:any namespace="##other" processContents="lax" minOccurs="0"
4997 maxOccurs="unbounded"/>
4998     </xsd:sequence>
4999     </xsd:complexType>
5000     </xsd:element>
5001     <!-- Filter allow list event by eventId or other params such as status and
5002 event type -->
5003     <xsd:complexType name="tTaskHistoryFilter">
5004     <xsd:choice>
5005     <xsd:element name="eventId" type="xsd:integer"/>
5006     <!-- Filter to allow narrow down query by status, principal, event
5007 Type. -->
5008     <xsd:sequence>
5009     <xsd:element name="status" type="tStatus" minOccurs="0"
5010 maxOccurs="unbounded"/>
5011     <xsd:element name="eventType" type="tTaskEventType" minOccurs="0"
5012 maxOccurs="unbounded"/>
5013     <xsd:element name="principal" type="xsd:string" minOccurs="0"/>
5014     <xsd:element name="afterEventTime" type="xsd:dateTime"
5015 minOccurs="0"/>
5016     <xsd:element name="beforeEventTime" type="xsd:dateTime"
5017 minOccurs="0"/>
5018     </xsd:sequence>
5019     </xsd:choice>
5020     </xsd:complexType>
5021 </xsd:schema>

```

5022

## D. WS-HumanTask Client API Port Type

```
5023 <?xml version="1.0" encoding="UTF-8"?>
5024 <!--
5025 Copyright (c) OASIS Open 2009. All Rights Reserved.
5026 -->
5027 <wsdl:definitions
5028   targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
5029 humantask/api/200803"
5030   xmlns="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/api/200803"
5031   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
5032   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
5033   xmlns:htt="http://docs.oasis-open.org/ns/bpel4people/ws-
5034 humantask/types/200803">
5035
5036   <wsdl:documentation>
5037     Web Service Definition for WS-HumanTask 1.1 - Operations for Client
5038 Applications
5039   </wsdl:documentation>
5040
5041   <wsdl:types>
5042     <xsd:schema
5043       targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
5044 humantask/api/200803"
5045       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
5046       xmlns:htt="http://docs.oasis-open.org/ns/bpel4people/ws-
5047 humantask/types/200803"
5048       elementFormDefault="qualified"
5049       blockDefault="#all">
5050
5051       <xsd:import
5052         namespace="http://docs.oasis-open.org/ns/bpel4people/ws-
5053 humantask/types/200803"
5054         schemaLocation="ws-humantask-types.xsd"/>
5055
5056       <!-- Input and output elements -->
5057       <xsd:element name="addAttachment">
5058         <xsd:complexType>
5059           <xsd:sequence>
5060             <xsd:element name="taskIdIdentifier" type="xsd:anyURI"/>
5061             <xsd:element name="name" type="xsd:string"/>
5062             <xsd:element name="accessType" type="xsd:string"/>
5063             <xsd:element name="contentType" type="xsd:string"/>
5064             <xsd:element name="attachment" type="xsd:anyType"/>
5065           </xsd:sequence>
5066         </xsd:complexType>
5067       </xsd:element>
5068       <xsd:element name="addAttachmentResponse">
5069         <xsd:complexType>
5070           <xsd:sequence>
5071             <xsd:element name="identifier" type="xsd:anyURI"/>
5072           </xsd:sequence>
5073         </xsd:complexType>
5074       </xsd:element>
5075
```

```

5076 <xsd:element name="addComment">
5077   <xsd:complexType>
5078     <xsd:sequence>
5079       <xsd:element name="identifier" type="xsd:anyURI"/>
5080       <xsd:element name="text" type="xsd:string"/>
5081     </xsd:sequence>
5082   </xsd:complexType>
5083 </xsd:element>
5084 <xsd:element name="addCommentResponse">
5085   <xsd:complexType>
5086     <xsd:sequence>
5087       <xsd:element name="commentID" type="xsd:anyURI"/>
5088     </xsd:sequence>
5089   </xsd:complexType>
5090 </xsd:element>
5091
5092 <xsd:element name="claim">
5093   <xsd:complexType>
5094     <xsd:sequence>
5095       <xsd:element name="identifier" type="xsd:anyURI"/>
5096     </xsd:sequence>
5097   </xsd:complexType>
5098 </xsd:element>
5099 <xsd:element name="claimResponse">
5100   <xsd:complexType>
5101     <xsd:sequence>
5102       <xsd:annotation>
5103         <xsd:documentation>Empty message</xsd:documentation>
5104       </xsd:annotation>
5105     </xsd:sequence>
5106   </xsd:complexType>
5107 </xsd:element>
5108
5109 <xsd:element name="batchClaim">
5110   <xsd:complexType>
5111     <xsd:sequence>
5112       <xsd:element name="identifier" type="xsd:anyURI"
5113 maxOccurs="unbounded"/>
5114     </xsd:sequence>
5115   </xsd:complexType>
5116 </xsd:element>
5117 <xsd:element name="batchClaimResponse">
5118   <xsd:complexType>
5119     <xsd:sequence>
5120       <xsd:element name="batchResponse" type="tBatchResponse"
5121 minOccurs="0" maxOccurs="unbounded"/>
5122     </xsd:sequence>
5123   </xsd:complexType>
5124 </xsd:element>
5125
5126 <xsd:element name="complete">
5127   <xsd:complexType>
5128     <xsd:sequence>
5129       <xsd:element name="identifier" type="xsd:anyURI"/>
5130       <xsd:element name="taskData" type="xsd:anyType" minOccurs="0"/>
5131     </xsd:sequence>
5132   </xsd:complexType>
5133 </xsd:element>

```

```

5134 <xsd:element name="completeResponse">
5135   <xsd:complexType>
5136     <xsd:sequence>
5137       <xsd:annotation>
5138         <xsd:documentation>Empty message</xsd:documentation>
5139       </xsd:annotation>
5140     </xsd:sequence>
5141   </xsd:complexType>
5142 </xsd:element>
5143
5144 <xsd:element name="batchComplete">
5145   <xsd:complexType>
5146     <xsd:sequence>
5147       <xsd:element name="identifier" type="xsd:anyURI"
5148 maxOccurs="unbounded"/>
5149       <xsd:element name="taskData" type="xsd:anyType" minOccurs="0"/>
5150     </xsd:sequence>
5151   </xsd:complexType>
5152 </xsd:element>
5153 <xsd:element name="batchCompleteResponse">
5154   <xsd:complexType>
5155     <xsd:sequence>
5156       <xsd:element name="batchResponse" type="tBatchResponse"
5157 minOccurs="0" maxOccurs="unbounded"/>
5158     </xsd:sequence>
5159   </xsd:complexType>
5160 </xsd:element>
5161
5162 <xsd:element name="delegate">
5163   <xsd:complexType>
5164     <xsd:sequence>
5165       <xsd:element name="identifier" type="xsd:anyURI"/>
5166       <xsd:element name="organizationalEntity"
5167 type="htt:tOrganizationalEntity"/>
5168     </xsd:sequence>
5169   </xsd:complexType>
5170 </xsd:element>
5171 <xsd:element name="delegateResponse">
5172   <xsd:complexType>
5173     <xsd:sequence>
5174       <xsd:annotation>
5175         <xsd:documentation>Empty message</xsd:documentation>
5176       </xsd:annotation>
5177     </xsd:sequence>
5178   </xsd:complexType>
5179 </xsd:element>
5180
5181 <xsd:element name="batchDelegate">
5182   <xsd:complexType>
5183     <xsd:sequence>
5184       <xsd:element name="identifier" type="xsd:anyURI"
5185 maxOccurs="unbounded"/>
5186       <xsd:element name="organizationalEntity"
5187 type="htt:tOrganizationalEntity"/>
5188     </xsd:sequence>
5189   </xsd:complexType>
5190 </xsd:element>
5191 <xsd:element name="batchDelegateResponse">

```

```

5192     <xsd:complexType>
5193         <xsd:sequence>
5194             <xsd:element name="batchResponse" type="tBatchResponse"
5195 minOccurs="0" maxOccurs="unbounded"/>
5196         </xsd:sequence>
5197     </xsd:complexType>
5198 </xsd:element>
5199
5200 <xsd:element name="deleteAttachment">
5201     <xsd:complexType>
5202         <xsd:sequence>
5203             <xsd:element name="taskIdentifier" type="xsd:anyURI"/>
5204             <xsd:element name="attachmentIdentifier" type="xsd:anyURI"/>
5205         </xsd:sequence>
5206     </xsd:complexType>
5207 </xsd:element>
5208 <xsd:element name="deleteAttachmentResponse">
5209     <xsd:complexType>
5210         <xsd:sequence>
5211             <xsd:annotation>
5212                 <xsd:documentation>Empty message</xsd:documentation>
5213             </xsd:annotation>
5214         </xsd:sequence>
5215     </xsd:complexType>
5216 </xsd:element>
5217
5218 <xsd:element name="deleteComment">
5219     <xsd:complexType>
5220         <xsd:sequence>
5221             <xsd:element name="taskIdentifier" type="xsd:anyURI"/>
5222             <xsd:element name="commentIdentifier" type="xsd:anyURI"/>
5223         </xsd:sequence>
5224     </xsd:complexType>
5225 </xsd:element>
5226 <xsd:element name="deleteCommentResponse">
5227     <xsd:complexType>
5228         <xsd:sequence>
5229             <xsd:annotation>
5230                 <xsd:documentation>Empty message</xsd:documentation>
5231             </xsd:annotation>
5232         </xsd:sequence>
5233     </xsd:complexType>
5234 </xsd:element>
5235
5236 <xsd:element name="deleteFault">
5237     <xsd:complexType>
5238         <xsd:sequence>
5239             <xsd:element name="identifier" type="xsd:anyURI"/>
5240         </xsd:sequence>
5241     </xsd:complexType>
5242 </xsd:element>
5243 <xsd:element name="deleteFaultResponse">
5244     <xsd:complexType>
5245         <xsd:sequence>
5246             <xsd:annotation>
5247                 <xsd:documentation>Empty message</xsd:documentation>
5248             </xsd:annotation>
5249         </xsd:sequence>

```

```

5250     </xsd:complexType>
5251 </xsd:element>
5252
5253 <xsd:element name="deleteOutput">
5254   <xsd:complexType>
5255     <xsd:sequence>
5256       <xsd:element name="identifier" type="xsd:anyURI"/>
5257     </xsd:sequence>
5258   </xsd:complexType>
5259 </xsd:element>
5260 <xsd:element name="deleteOutputResponse">
5261   <xsd:complexType>
5262     <xsd:sequence>
5263       <xsd:annotation>
5264         <xsd:documentation>Empty message</xsd:documentation>
5265       </xsd:annotation>
5266     </xsd:sequence>
5267   </xsd:complexType>
5268 </xsd:element>
5269
5270 <xsd:element name="fail">
5271   <xsd:complexType>
5272     <xsd:sequence>
5273       <xsd:element name="identifier" type="xsd:anyURI"/>
5274       <xsd:element name="fault" type="htt:tFault" minOccurs="0"/>
5275     </xsd:sequence>
5276   </xsd:complexType>
5277 </xsd:element>
5278 <xsd:element name="failResponse">
5279   <xsd:complexType>
5280     <xsd:sequence>
5281       <xsd:annotation>
5282         <xsd:documentation>Empty message</xsd:documentation>
5283       </xsd:annotation>
5284     </xsd:sequence>
5285   </xsd:complexType>
5286 </xsd:element>
5287
5288 <xsd:element name="batchFail">
5289   <xsd:complexType>
5290     <xsd:sequence>
5291       <xsd:element name="identifier" type="xsd:anyURI"
5292 maxOccurs="unbounded"/>
5293       <xsd:element name="fault" type="htt:tFault" minOccurs="0"/>
5294     </xsd:sequence>
5295   </xsd:complexType>
5296 </xsd:element>
5297 <xsd:element name="batchFailResponse">
5298   <xsd:complexType>
5299     <xsd:sequence>
5300       <xsd:element name="batchResponse" type="tBatchResponse"
5301 minOccurs="0" maxOccurs="unbounded"/>
5302     </xsd:sequence>
5303   </xsd:complexType>
5304 </xsd:element>
5305
5306 <xsd:element name="forward">
5307   <xsd:complexType>

```

```

5308     <xsd:sequence>
5309         <xsd:element name="identifier" type="xsd:anyURI"/>
5310         <xsd:element name="organizationalEntity"
5311 type="htt:tOrganizationalEntity"/>
5312     </xsd:sequence>
5313 </xsd:complexType>
5314 </xsd:element>
5315 <xsd:element name="forwardResponse">
5316     <xsd:complexType>
5317         <xsd:sequence>
5318             <xsd:annotation>
5319                 <xsd:documentation>Empty message</xsd:documentation>
5320             </xsd:annotation>
5321         </xsd:sequence>
5322     </xsd:complexType>
5323 </xsd:element>
5324
5325     <xsd:element name="batchForward">
5326         <xsd:complexType>
5327             <xsd:sequence>
5328                 <xsd:element name="identifier" type="xsd:anyURI"
5329 maxOccurs="unbounded"/>
5330                 <xsd:element name="organizationalEntity"
5331 type="htt:tOrganizationalEntity"/>
5332             </xsd:sequence>
5333         </xsd:complexType>
5334     </xsd:element>
5335     <xsd:element name="batchForwardResponse">
5336         <xsd:complexType>
5337             <xsd:sequence>
5338                 <xsd:element name="batchResponse" type="tBatchResponse"
5339 minOccurs="0" maxOccurs="unbounded"/>
5340             </xsd:sequence>
5341         </xsd:complexType>
5342     </xsd:element>
5343
5344     <xsd:element name="getAttachment">
5345         <xsd:complexType>
5346             <xsd:sequence>
5347                 <xsd:element name="taskIdIdentifier" type="xsd:anyURI"/>
5348                 <xsd:element name="attachmentIdentifier" type="xsd:anyURI"/>
5349             </xsd:sequence>
5350         </xsd:complexType>
5351     </xsd:element>
5352     <xsd:element name="getAttachmentResponse">
5353         <xsd:complexType>
5354             <xsd:sequence>
5355                 <xsd:element name="attachment" type="htt:tAttachment"
5356 minOccurs="0" maxOccurs="unbounded"/>
5357             </xsd:sequence>
5358         </xsd:complexType>
5359     </xsd:element>
5360
5361     <xsd:element name="getAttachmentInfos">
5362         <xsd:complexType>
5363             <xsd:sequence>
5364                 <xsd:element name="identifier" type="xsd:anyURI"/>
5365             </xsd:sequence>

```

```

5366     </xsd:complexType>
5367 </xsd:element>
5368 <xsd:element name="getAttachmentInfosResponse">
5369   <xsd:complexType>
5370     <xsd:sequence>
5371       <xsd:element name="info" type="htt:tAttachmentInfo" minOccurs="0"
5372 maxOccurs="unbounded"/>
5373     </xsd:sequence>
5374   </xsd:complexType>
5375 </xsd:element>
5376
5377 <xsd:element name="getComments">
5378   <xsd:complexType>
5379     <xsd:sequence>
5380       <xsd:element name="identifier" type="xsd:anyURI"/>
5381     </xsd:sequence>
5382   </xsd:complexType>
5383 </xsd:element>
5384 <xsd:element name="getCommentsResponse">
5385   <xsd:complexType>
5386     <xsd:sequence>
5387       <xsd:element name="comment" type="htt:tComment" minOccurs="0"
5388 maxOccurs="unbounded"/>
5389     </xsd:sequence>
5390   </xsd:complexType>
5391 </xsd:element>
5392
5393 <xsd:element name="getFault">
5394   <xsd:complexType>
5395     <xsd:sequence>
5396       <xsd:element name="identifier" type="xsd:anyURI"/>
5397     </xsd:sequence>
5398   </xsd:complexType>
5399 </xsd:element>
5400 <xsd:element name="getFaultResponse">
5401   <xsd:complexType>
5402     <xsd:sequence>
5403       <xsd:element name="fault" type="htt:tFault"/>
5404     </xsd:sequence>
5405   </xsd:complexType>
5406 </xsd:element>
5407
5408 <xsd:element name="getInput">
5409   <xsd:complexType>
5410     <xsd:sequence>
5411       <xsd:element name="identifier" type="xsd:anyURI"/>
5412       <xsd:element name="part" type="xsd:NCName" minOccurs="0"/>
5413     </xsd:sequence>
5414   </xsd:complexType>
5415 </xsd:element>
5416 <xsd:element name="getInputResponse">
5417   <xsd:complexType>
5418     <xsd:sequence>
5419       <xsd:element name="taskData" type="xsd:anyType"/>
5420     </xsd:sequence>
5421   </xsd:complexType>
5422 </xsd:element>
5423

```

```

5424 <xsd:element name="getOutcome">
5425   <xsd:complexType>
5426     <xsd:sequence>
5427       <xsd:element name="identifier" type="xsd:anyURI"/>
5428     </xsd:sequence>
5429   </xsd:complexType>
5430 </xsd:element>
5431 <xsd:element name="getOutcomeResponse">
5432   <xsd:complexType>
5433     <xsd:sequence>
5434       <xsd:element name="outcome" type="xsd:string"/>
5435     </xsd:sequence>
5436   </xsd:complexType>
5437 </xsd:element>
5438
5439 <xsd:element name="getOutput">
5440   <xsd:complexType>
5441     <xsd:sequence>
5442       <xsd:element name="identifier" type="xsd:anyURI"/>
5443       <xsd:element name="part" type="xsd:NCName" minOccurs="0"/>
5444     </xsd:sequence>
5445   </xsd:complexType>
5446 </xsd:element>
5447 <xsd:element name="getOutputResponse">
5448   <xsd:complexType>
5449     <xsd:sequence>
5450       <xsd:element name="taskData" type="xsd:anyType"/>
5451     </xsd:sequence>
5452   </xsd:complexType>
5453 </xsd:element>
5454
5455 <xsd:element name="getParentTask">
5456   <xsd:complexType>
5457     <xsd:sequence>
5458       <xsd:element name="taskIdentifier" type="xsd:anyURI"/>
5459     </xsd:sequence>
5460   </xsd:complexType>
5461 </xsd:element>
5462 <xsd:element name="getParentTaskResponse">
5463   <xsd:complexType>
5464     <xsd:sequence>
5465       <xsd:element name="parentTask" type="htt:tTask" minOccurs="0"/>
5466     </xsd:sequence>
5467   </xsd:complexType>
5468 </xsd:element>
5469
5470 <xsd:element name="getParentTaskIdentifier">
5471   <xsd:complexType>
5472     <xsd:sequence>
5473       <xsd:element name="taskIdentifier" type="xsd:anyURI"/>
5474     </xsd:sequence>
5475   </xsd:complexType>
5476 </xsd:element>
5477 <xsd:element name="getParentTaskIdentifierResponse">
5478   <xsd:complexType>
5479     <xsd:sequence>
5480       <xsd:element name="parentTaskIdentifier" type="xsd:anyURI"
5481 minOccurs="0"/>

```

```

5482     </xsd:sequence>
5483     </xsd:complexType>
5484 </xsd:element>
5485
5486 <xsd:element name="getRendering">
5487     <xsd:complexType>
5488         <xsd:sequence>
5489             <xsd:element name="identifier" type="xsd:anyURI"/>
5490             <xsd:element name="renderingType" type="xsd:QName"/>
5491         </xsd:sequence>
5492     </xsd:complexType>
5493 </xsd:element>
5494 <xsd:element name="getRenderingResponse">
5495     <xsd:complexType>
5496         <xsd:sequence>
5497             <xsd:element name="rendering" type="xsd:anyType"/>
5498         </xsd:sequence>
5499     </xsd:complexType>
5500 </xsd:element>
5501
5502 <xsd:element name="getRenderingTypes">
5503     <xsd:complexType>
5504         <xsd:sequence>
5505             <xsd:element name="identifier" type="xsd:anyURI"/>
5506         </xsd:sequence>
5507     </xsd:complexType>
5508 </xsd:element>
5509 <xsd:element name="getRenderingTypesResponse">
5510     <xsd:complexType>
5511         <xsd:sequence>
5512             <xsd:element name="renderingType" type="xsd:QName" minOccurs="0"
5513 maxOccurs="unbounded"/>
5514         </xsd:sequence>
5515     </xsd:complexType>
5516 </xsd:element>
5517
5518 <xsd:element name="getSubtaskIdentifiers">
5519     <xsd:complexType>
5520         <xsd:sequence>
5521             <xsd:element name="taskIdentifier" type="xsd:anyURI"/>
5522         </xsd:sequence>
5523     </xsd:complexType>
5524 </xsd:element>
5525 <xsd:element name="getSubtaskIdentifiersResponse">
5526     <xsd:complexType>
5527         <xsd:sequence>
5528             <xsd:element name="subtaskIdentifier" type="xsd:anyURI"
5529 minOccurs="0" maxOccurs="unbounded"/>
5530         </xsd:sequence>
5531     </xsd:complexType>
5532 </xsd:element>
5533
5534 <xsd:element name="getSubtasks">
5535     <xsd:complexType>
5536         <xsd:sequence>
5537             <xsd:element name="taskIdentifier" type="xsd:anyURI"/>
5538         </xsd:sequence>
5539     </xsd:complexType>

```

```

5540     </xsd:element>
5541     <xsd:element name="getSubtasksResponse">
5542         <xsd:complexType>
5543             <xsd:sequence>
5544                 <xsd:element name="subtask" type="htt:tTask" minOccurs="0"
5545 minOccurs="unbounded"/>
5546             </xsd:sequence>
5547         </xsd:complexType>
5548     </xsd:element>
5549
5550     <xsd:element name="getTaskDescription">
5551         <xsd:complexType>
5552             <xsd:sequence>
5553                 <xsd:element name="identifier" type="xsd:anyURI"/>
5554                 <xsd:element name="contentType" type="xsd:string" minOccurs="0"/>
5555             </xsd:sequence>
5556         </xsd:complexType>
5557     </xsd:element>
5558     <xsd:element name="getTaskDescriptionResponse">
5559         <xsd:complexType>
5560             <xsd:sequence>
5561                 <xsd:element name="description" type="xsd:string"/>
5562             </xsd:sequence>
5563         </xsd:complexType>
5564     </xsd:element>
5565
5566     <xsd:element name="getTaskDetails">
5567         <xsd:complexType>
5568             <xsd:sequence>
5569                 <xsd:element name="identifier" type="xsd:anyURI"/>
5570             </xsd:sequence>
5571         </xsd:complexType>
5572     </xsd:element>
5573     <xsd:element name="getTaskDetailsResponse">
5574         <xsd:complexType>
5575             <xsd:sequence>
5576                 <xsd:element name="taskDetails" type="htt:tTaskDetails"/>
5577             </xsd:sequence>
5578         </xsd:complexType>
5579     </xsd:element>
5580
5581     <xsd:element name="getTaskHistory">
5582         <xsd:complexType>
5583             <xsd:sequence>
5584                 <xsd:element name="identifier" type="xsd:anyURI"/>
5585                 <xsd:element name="filter" type="htt:tTaskHistoryFilter"
5586 minOccurs="0"/>
5587                 <xsd:element name="startIndex" type="xsd:int" minOccurs="0"/>
5588                 <xsd:element name="maxTasks" type="xsd:int" minOccurs="0"/>
5589             </xsd:sequence>
5590             <xsd:attribute name="includeData" type="xsd:boolean"/>
5591         </xsd:complexType>
5592     </xsd:element>
5593     <xsd:element name="getTaskHistoryResponse">
5594         <xsd:complexType>
5595             <xsd:sequence>
5596                 <xsd:element name="taskEvent" type="htt:tTaskEventType"
5597 minOccurs="0" maxOccurs="unbounded"/>

```

```

5598     </xsd:sequence>
5599   </xsd:complexType>
5600 </xsd:element>
5601
5602   <xsd:element name="getTaskInstanceData">
5603     <xsd:complexType>
5604       <xsd:sequence>
5605         <xsd:element name="identifier" type="xsd:anyURI"/>
5606         <xsd:element name="properties" type="xsd:string"/>
5607         <xsd:element name="renderingPreferences"
5608 type="htt:tRenderingTypes" minOccurs="0" maxOccurs="unbounded"/>
5609       </xsd:sequence>
5610     </xsd:complexType>
5611   </xsd:element>
5612   <xsd:element name="getTaskInstanceDataResponse">
5613     <xsd:complexType>
5614       <xsd:sequence>
5615         <xsd:element name="taskInstanceData"
5616 type="htt:tTaskInstanceData"/>
5617       </xsd:sequence>
5618     </xsd:complexType>
5619   </xsd:element>
5620
5621   <xsd:element name="getTaskOperations">
5622     <xsd:complexType>
5623       <xsd:sequence>
5624         <xsd:element name="identifier" type="xsd:anyURI"/>
5625       </xsd:sequence>
5626     </xsd:complexType>
5627   </xsd:element>
5628   <xsd:element name="getTaskOperationsResponse">
5629     <xsd:complexType>
5630       <xsd:sequence>
5631         <xsd:element name="taskOperations" type="htt:tTaskOperations"/>
5632       </xsd:sequence>
5633     </xsd:complexType>
5634   </xsd:element>
5635
5636   <xsd:element name="hasSubtasks">
5637     <xsd:complexType>
5638       <xsd:sequence>
5639         <xsd:element name="taskIdentifier" type="xsd:anyURI"/>
5640       </xsd:sequence>
5641     </xsd:complexType>
5642   </xsd:element>
5643   <xsd:element name="hasSubtasksResponse">
5644     <xsd:complexType>
5645       <xsd:sequence>
5646         <xsd:element name="result" type="xsd:boolean"/>
5647       </xsd:sequence>
5648     </xsd:complexType>
5649   </xsd:element>
5650
5651   <xsd:element name="instantiateSubtask">
5652     <xsd:complexType>
5653       <xsd:sequence>
5654         <xsd:element name="taskIdentifier" type="xsd:anyURI"/>
5655         <xsd:element name="name" type="xsd:string"/>

```

```

5656     </xsd:sequence>
5657     </xsd:complexType>
5658 </xsd:element>
5659 <xsd:element name="instantiateSubtaskResponse">
5660     <xsd:complexType>
5661     <xsd:sequence>
5662     <xsd:element name="subtaskIdentifier" type="xsd:anyURI"/>
5663     </xsd:sequence>
5664     </xsd:complexType>
5665 </xsd:element>
5666
5667 <xsd:element name="isSubtask">
5668     <xsd:complexType>
5669     <xsd:sequence>
5670     <xsd:element name="taskIdentifier" type="xsd:anyURI"/>
5671     </xsd:sequence>
5672     </xsd:complexType>
5673 </xsd:element>
5674 <xsd:element name="isSubtaskResponse">
5675     <xsd:complexType>
5676     <xsd:sequence>
5677     <xsd:element name="result" type="xsd:boolean"/>
5678     </xsd:sequence>
5679     </xsd:complexType>
5680 </xsd:element>
5681
5682 <xsd:element name="release">
5683     <xsd:complexType>
5684     <xsd:sequence>
5685     <xsd:element name="identifier" type="xsd:anyURI"/>
5686     </xsd:sequence>
5687     </xsd:complexType>
5688 </xsd:element>
5689 <xsd:element name="releaseResponse">
5690     <xsd:complexType>
5691     <xsd:sequence>
5692     <xsd:annotation>
5693     <xsd:documentation>Empty message</xsd:documentation>
5694     </xsd:annotation>
5695     </xsd:sequence>
5696     </xsd:complexType>
5697 </xsd:element>
5698
5699 <xsd:element name="batchRelease">
5700     <xsd:complexType>
5701     <xsd:sequence>
5702     <xsd:element name="identifier" type="xsd:anyURI"
5703 maxOccurs="unbounded"/>
5704     </xsd:sequence>
5705     </xsd:complexType>
5706 </xsd:element>
5707 <xsd:element name="batchReleaseResponse">
5708     <xsd:complexType>
5709     <xsd:sequence>
5710     <xsd:element name="batchResponse" type="tBatchResponse"
5711 minOccurs="0" maxOccurs="unbounded"/>
5712     </xsd:sequence>
5713 </xsd:complexType>

```

```

5714 </xsd:element>
5715
5716 <xsd:element name="remove">
5717   <xsd:complexType>
5718     <xsd:sequence>
5719       <xsd:element name="identifier" type="xsd:anyURI"/>
5720     </xsd:sequence>
5721   </xsd:complexType>
5722 </xsd:element>
5723 <xsd:element name="removeResponse">
5724   <xsd:complexType>
5725     <xsd:sequence>
5726       <xsd:annotation>
5727         <xsd:documentation>Empty message</xsd:documentation>
5728       </xsd:annotation>
5729     </xsd:sequence>
5730   </xsd:complexType>
5731 </xsd:element>
5732
5733 <xsd:element name="batchRemove">
5734   <xsd:complexType>
5735     <xsd:sequence>
5736       <xsd:element name="identifier" type="xsd:anyURI"
5737 maxOccurs="unbounded"/>
5738     </xsd:sequence>
5739   </xsd:complexType>
5740 </xsd:element>
5741 <xsd:element name="batchRemoveResponse">
5742   <xsd:complexType>
5743     <xsd:sequence>
5744       <xsd:element name="batchResponse" type="tBatchResponse"
5745 minOccurs="0" maxOccurs="unbounded"/>
5746     </xsd:sequence>
5747   </xsd:complexType>
5748 </xsd:element>
5749
5750 <xsd:element name="resume">
5751   <xsd:complexType>
5752     <xsd:sequence>
5753       <xsd:element name="identifier" type="xsd:anyURI"/>
5754     </xsd:sequence>
5755   </xsd:complexType>
5756 </xsd:element>
5757 <xsd:element name="resumeResponse">
5758   <xsd:complexType>
5759     <xsd:sequence>
5760       <xsd:annotation>
5761         <xsd:documentation>Empty message</xsd:documentation>
5762       </xsd:annotation>
5763     </xsd:sequence>
5764   </xsd:complexType>
5765 </xsd:element>
5766
5767 <xsd:element name="batchResume">
5768   <xsd:complexType>
5769     <xsd:sequence>
5770       <xsd:element name="identifier" type="xsd:anyURI"
5771 maxOccurs="unbounded"/>

```

```

5772     </xsd:sequence>
5773     </xsd:complexType>
5774 </xsd:element>
5775 <xsd:element name="batchResumeResponse">
5776     <xsd:complexType>
5777         <xsd:sequence>
5778             <xsd:element name="batchResponse" type="tBatchResponse"
5779 minOccurs="0" maxOccurs="unbounded"/>
5780         </xsd:sequence>
5781     </xsd:complexType>
5782 </xsd:element>
5783
5784 <xsd:element name="setFault">
5785     <xsd:complexType>
5786         <xsd:sequence>
5787             <xsd:element name="identifier" type="xsd:anyURI"/>
5788             <xsd:element name="fault" type="htt:tFault"/>
5789         </xsd:sequence>
5790     </xsd:complexType>
5791 </xsd:element>
5792 <xsd:element name="setFaultResponse">
5793     <xsd:complexType>
5794         <xsd:sequence>
5795             <xsd:annotation>
5796                 <xsd:documentation>Empty message</xsd:documentation>
5797             </xsd:annotation>
5798         </xsd:sequence>
5799     </xsd:complexType>
5800 </xsd:element>
5801
5802 <xsd:element name="setOutput">
5803     <xsd:complexType>
5804         <xsd:sequence>
5805             <xsd:element name="identifier" type="xsd:anyURI"/>
5806             <xsd:element name="part" type="xsd:NCName" minOccurs="0"/>
5807             <xsd:element name="taskData" type="xsd:anyType"/>
5808         </xsd:sequence>
5809     </xsd:complexType>
5810 </xsd:element>
5811 <xsd:element name="setOutputResponse">
5812     <xsd:complexType>
5813         <xsd:sequence>
5814             <xsd:annotation>
5815                 <xsd:documentation>Empty message</xsd:documentation>
5816             </xsd:annotation>
5817         </xsd:sequence>
5818     </xsd:complexType>
5819 </xsd:element>
5820
5821 <xsd:element name="setPriority">
5822     <xsd:complexType>
5823         <xsd:sequence>
5824             <xsd:element name="identifier" type="xsd:anyURI"/>
5825             <xsd:element name="priority" type="htt:tPriority"/>
5826         </xsd:sequence>
5827     </xsd:complexType>
5828 </xsd:element>
5829 <xsd:element name="setPriorityResponse">

```

```

5830     <xsd:complexType>
5831         <xsd:sequence>
5832             <xsd:annotation>
5833                 <xsd:documentation>Empty message</xsd:documentation>
5834             </xsd:annotation>
5835         </xsd:sequence>
5836     </xsd:complexType>
5837 </xsd:element>
5838
5839     <xsd:element name="batchSetPriority">
5840         <xsd:complexType>
5841             <xsd:sequence>
5842                 <xsd:element name="identifier" type="xsd:anyURI"
5843 maxOccurs="unbounded"/>
5844                 <xsd:element name="priority" type="htt:tPriority"/>
5845             </xsd:sequence>
5846         </xsd:complexType>
5847     </xsd:element>
5848     <xsd:element name="batchSetPriorityResponse">
5849         <xsd:complexType>
5850             <xsd:sequence>
5851                 <xsd:element name="batchResponse" type="tBatchResponse"
5852 minOccurs="0" maxOccurs="unbounded"/>
5853             </xsd:sequence>
5854         </xsd:complexType>
5855     </xsd:element>
5856
5857     <xsd:element name="setTaskCompletionDeadlineExpression">
5858         <xsd:complexType>
5859             <xsd:sequence>
5860                 <xsd:element name="identifier" type="xsd:anyURI"/>
5861                 <xsd:element name="deadlineName" type="xsd:NCName"/>
5862                 <xsd:element name="deadlineExpression" type="xsd:string"/>
5863             </xsd:sequence>
5864         </xsd:complexType>
5865     </xsd:element>
5866     <xsd:element name="setTaskCompletionDeadlineExpressionResponse">
5867         <xsd:complexType>
5868             <xsd:sequence>
5869                 <xsd:annotation>
5870                     <xsd:documentation>Empty message</xsd:documentation>
5871                 </xsd:annotation>
5872             </xsd:sequence>
5873         </xsd:complexType>
5874     </xsd:element>
5875
5876     <xsd:element name="setTaskCompletionDurationExpression">
5877         <xsd:complexType>
5878             <xsd:sequence>
5879                 <xsd:element name="identifier" type="xsd:anyURI"/>
5880                 <xsd:element name="deadlineName" type="xsd:NCName"/>
5881                 <xsd:element name="durationExpression" type="xsd:string"/>
5882             </xsd:sequence>
5883         </xsd:complexType>
5884     </xsd:element>
5885     <xsd:element name="setTaskCompletionDurationExpressionResponse">
5886         <xsd:complexType>
5887             <xsd:sequence>

```

```

5888     <xsd:annotation>
5889         <xsd:documentation>Empty message</xsd:documentation>
5890     </xsd:annotation>
5891 </xsd:sequence>
5892 </xsd:complexType>
5893 </xsd:element>
5894
5895 <xsd:element name="setTaskStartDeadlineExpression">
5896     <xsd:complexType>
5897         <xsd:sequence>
5898             <xsd:element name="identifier" type="xsd:anyURI"/>
5899             <xsd:element name="deadlineName" type="xsd:NCName"/>
5900             <xsd:element name="deadlineExpression" type="xsd:string"/>
5901         </xsd:sequence>
5902     </xsd:complexType>
5903 </xsd:element>
5904 <xsd:element name="setTaskStartDeadlineExpressionResponse">
5905     <xsd:complexType>
5906         <xsd:sequence>
5907             <xsd:annotation>
5908                 <xsd:documentation>Empty message</xsd:documentation>
5909             </xsd:annotation>
5910         </xsd:sequence>
5911     </xsd:complexType>
5912 </xsd:element>
5913
5914 <xsd:element name="setTaskStartDurationExpression">
5915     <xsd:complexType>
5916         <xsd:sequence>
5917             <xsd:element name="identifier" type="xsd:anyURI"/>
5918             <xsd:element name="deadlineName" type="xsd:NCName"/>
5919             <xsd:element name="durationExpression" type="xsd:string"/>
5920         </xsd:sequence>
5921     </xsd:complexType>
5922 </xsd:element>
5923 <xsd:element name="setTaskStartDurationExpressionResponse">
5924     <xsd:complexType>
5925         <xsd:sequence>
5926             <xsd:annotation>
5927                 <xsd:documentation>Empty message</xsd:documentation>
5928             </xsd:annotation>
5929         </xsd:sequence>
5930     </xsd:complexType>
5931 </xsd:element>
5932
5933 <xsd:element name="skip">
5934     <xsd:complexType>
5935         <xsd:sequence>
5936             <xsd:element name="identifier" type="xsd:anyURI"/>
5937         </xsd:sequence>
5938     </xsd:complexType>
5939 </xsd:element>
5940 <xsd:element name="skipResponse">
5941     <xsd:complexType>
5942         <xsd:sequence>
5943             <xsd:annotation>
5944                 <xsd:documentation>Empty message</xsd:documentation>
5945             </xsd:annotation>

```

```

5946     </xsd:sequence>
5947     </xsd:complexType>
5948 </xsd:element>
5949
5950     <xsd:element name="batchSkip">
5951         <xsd:complexType>
5952             <xsd:sequence>
5953                 <xsd:element name="identifier" type="xsd:anyURI"
5954 maxOccurs="unbounded"/>
5955             </xsd:sequence>
5956         </xsd:complexType>
5957     </xsd:element>
5958     <xsd:element name="batchSkipResponse">
5959         <xsd:complexType>
5960             <xsd:sequence>
5961                 <xsd:element name="batchResponse" type="tBatchResponse"
5962 minOccurs="0" maxOccurs="unbounded"/>
5963             </xsd:sequence>
5964         </xsd:complexType>
5965     </xsd:element>
5966
5967     <xsd:element name="start">
5968         <xsd:complexType>
5969             <xsd:sequence>
5970                 <xsd:element name="identifier" type="xsd:anyURI"/>
5971             </xsd:sequence>
5972         </xsd:complexType>
5973     </xsd:element>
5974     <xsd:element name="startResponse">
5975         <xsd:complexType>
5976             <xsd:sequence>
5977                 <xsd:annotation>
5978                     <xsd:documentation>Empty message</xsd:documentation>
5979                 </xsd:annotation>
5980             </xsd:sequence>
5981         </xsd:complexType>
5982     </xsd:element>
5983
5984     <xsd:element name="batchStart">
5985         <xsd:complexType>
5986             <xsd:sequence>
5987                 <xsd:element name="identifier" type="xsd:anyURI"
5988 maxOccurs="unbounded"/>
5989             </xsd:sequence>
5990         </xsd:complexType>
5991     </xsd:element>
5992     <xsd:element name="batchStartResponse">
5993         <xsd:complexType>
5994             <xsd:sequence>
5995                 <xsd:element name="batchResponse" type="tBatchResponse"
5996 minOccurs="0" maxOccurs="unbounded"/>
5997             </xsd:sequence>
5998         </xsd:complexType>
5999     </xsd:element>
6000
6001     <xsd:element name="stop">
6002         <xsd:complexType>
6003             <xsd:sequence>

```

```

6004         <xsd:element name="identifier" type="xsd:anyURI"/>
6005     </xsd:sequence>
6006 </xsd:complexType>
6007 </xsd:element>
6008 <xsd:element name="stopResponse">
6009     <xsd:complexType>
6010         <xsd:sequence>
6011             <xsd:annotation>
6012                 <xsd:documentation>Empty message</xsd:documentation>
6013             </xsd:annotation>
6014         </xsd:sequence>
6015     </xsd:complexType>
6016 </xsd:element>
6017
6018 <xsd:element name="batchStop">
6019     <xsd:complexType>
6020         <xsd:sequence>
6021             <xsd:element name="identifier" type="xsd:anyURI"
6022 maxOccurs="unbounded"/>
6023         </xsd:sequence>
6024     </xsd:complexType>
6025 </xsd:element>
6026 <xsd:element name="batchStopResponse">
6027     <xsd:complexType>
6028         <xsd:sequence>
6029             <xsd:element name="batchResponse" type="tBatchResponse"
6030 minOccurs="0" maxOccurs="unbounded"/>
6031         </xsd:sequence>
6032     </xsd:complexType>
6033 </xsd:element>
6034
6035 <xsd:element name="suspend">
6036     <xsd:complexType>
6037         <xsd:sequence>
6038             <xsd:element name="identifier" type="xsd:anyURI"/>
6039         </xsd:sequence>
6040     </xsd:complexType>
6041 </xsd:element>
6042 <xsd:element name="suspendResponse">
6043     <xsd:complexType>
6044         <xsd:sequence>
6045             <xsd:annotation>
6046                 <xsd:documentation>Empty message</xsd:documentation>
6047             </xsd:annotation>
6048         </xsd:sequence>
6049     </xsd:complexType>
6050 </xsd:element>
6051
6052 <xsd:element name="batchSuspend">
6053     <xsd:complexType>
6054         <xsd:sequence>
6055             <xsd:element name="identifier" type="xsd:anyURI"
6056 maxOccurs="unbounded"/>
6057         </xsd:sequence>
6058     </xsd:complexType>
6059 </xsd:element>
6060 <xsd:element name="batchSuspendResponse">
6061     <xsd:complexType>

```

```

6062     <xsd:sequence>
6063         <xsd:element name="batchResponse" type="tBatchResponse"
6064 minOccurs="0" maxOccurs="unbounded"/>
6065     </xsd:sequence>
6066 </xsd:complexType>
6067 </xsd:element>
6068
6069 <xsd:element name="suspendUntil">
6070 <xsd:complexType>
6071 <xsd:sequence>
6072 <xsd:element name="identifier" type="xsd:anyURI"/>
6073 <xsd:element name="time" type="htt:tTime"/>
6074 </xsd:sequence>
6075 </xsd:complexType>
6076 </xsd:element>
6077 <xsd:element name="suspendUntilResponse">
6078 <xsd:complexType>
6079 <xsd:sequence>
6080 <xsd:annotation>
6081 <xsd:documentation>Empty message</xsd:documentation>
6082 </xsd:annotation>
6083 </xsd:sequence>
6084 </xsd:complexType>
6085 </xsd:element>
6086
6087 <xsd:element name="batchSuspendUntil">
6088 <xsd:complexType>
6089 <xsd:sequence>
6090 <xsd:element name="identifier" type="xsd:anyURI"
6091 maxOccurs="unbounded"/>
6092 <xsd:element name="time" type="htt:tTime"/>
6093 </xsd:sequence>
6094 </xsd:complexType>
6095 </xsd:element>
6096 <xsd:element name="batchSuspendUntilResponse">
6097 <xsd:complexType>
6098 <xsd:sequence>
6099 <xsd:element name="batchResponse" type="tBatchResponse"
6100 minOccurs="0" maxOccurs="unbounded"/>
6101 </xsd:sequence>
6102 </xsd:complexType>
6103 </xsd:element>
6104
6105 <xsd:element name="updateComment">
6106 <xsd:complexType>
6107 <xsd:sequence>
6108 <xsd:element name="taskIdentifier" type="xsd:anyURI"/>
6109 <xsd:element name="commentIdentifier" type="xsd:anyURI"/>
6110 <xsd:element name="text" type="xsd:string"/>
6111 </xsd:sequence>
6112 </xsd:complexType>
6113 </xsd:element>
6114 <xsd:element name="updateCommentResponse">
6115 <xsd:complexType>
6116 <xsd:sequence>
6117 <xsd:annotation>
6118 <xsd:documentation>Empty message</xsd:documentation>
6119 </xsd:annotation>

```

```

6120     </xsd:sequence>
6121     </xsd:complexType>
6122 </xsd:element>
6123
6124     <xsd:element name="getMyTaskAbstracts">
6125         <xsd:complexType>
6126             <xsd:sequence>
6127                 <xsd:element name="taskType" type="xsd:string"/>
6128                 <xsd:element name="genericHumanRole" type="xsd:string"
6129 minOccurs="0"/>
6130                 <xsd:element name="workQueue" type="xsd:string" minOccurs="0"/>
6131                 <xsd:element name="status" type="htt:tStatus" minOccurs="0"
6132 maxOccurs="unbounded"/>
6133                 <xsd:element name="whereClause" type="xsd:string" minOccurs="0"/>
6134                 <xsd:element name="orderByClause" type="xsd:string"
6135 minOccurs="0"/>
6136                 <xsd:element name="createdOnClause" type="xsd:string"
6137 minOccurs="0"/>
6138                 <xsd:element name="maxTasks" type="xsd:int" minOccurs="0"/>
6139                 <xsd:element name="taskIndexOffset" type="xsd:int"
6140 minOccurs="0"/>
6141             </xsd:sequence>
6142         </xsd:complexType>
6143     </xsd:element>
6144     <xsd:element name="getMyTaskAbstractsResponse">
6145         <xsd:complexType>
6146             <xsd:sequence>
6147                 <xsd:element name="taskAbstract" type="htt:tTaskAbstract"
6148 minOccurs="0" maxOccurs="unbounded"/>
6149             </xsd:sequence>
6150         </xsd:complexType>
6151     </xsd:element>
6152
6153     <xsd:element name="getMyTaskDetails">
6154         <xsd:complexType>
6155             <xsd:sequence>
6156                 <xsd:element name="taskType" type="xsd:string"/>
6157                 <xsd:element name="genericHumanRole" type="xsd:string"
6158 minOccurs="0"/>
6159                 <xsd:element name="workQueue" type="xsd:string" minOccurs="0"/>
6160                 <xsd:element name="status" type="htt:tStatus" minOccurs="0"
6161 maxOccurs="unbounded"/>
6162                 <xsd:element name="whereClause" type="xsd:string" minOccurs="0"/>
6163                 <xsd:element name="orderByClause" type="xsd:string"
6164 minOccurs="0"/>
6165                 <xsd:element name="createdOnClause" type="xsd:string"
6166 minOccurs="0"/>
6167                 <xsd:element name="maxTasks" type="xsd:int" minOccurs="0"/>
6168                 <xsd:element name="taskIndexOffset" type="xsd:int"
6169 minOccurs="0"/>
6170             </xsd:sequence>
6171         </xsd:complexType>
6172     </xsd:element>
6173     <xsd:element name="getMyTaskDetailsResponse">
6174         <xsd:complexType>
6175             <xsd:sequence>
6176                 <xsd:element name="taskDetails" type="htt:tTaskDetails"
6177 minOccurs="0" maxOccurs="unbounded"/>

```

```

6178         </xsd:sequence>
6179     </xsd:complexType>
6180 </xsd:element>
6181
6182     <xsd:element name="query">
6183         <xsd:complexType>
6184             <xsd:sequence>
6185                 <xsd:element name="selectClause" type="xsd:string"/>
6186                 <xsd:element name="whereClause" type="xsd:string" minOccurs="0"/>
6187                 <xsd:element name="orderByClause" type="xsd:string"
6188 minOccurs="0"/>
6189                 <xsd:element name="maxTasks" type="xsd:int" minOccurs="0"/>
6190                 <xsd:element name="taskIndexOffset" type="xsd:int"
6191 minOccurs="0"/>
6192             </xsd:sequence>
6193         </xsd:complexType>
6194     </xsd:element>
6195     <xsd:element name="queryResponse">
6196         <xsd:complexType>
6197             <xsd:sequence>
6198                 <xsd:element name="taskQueryResultSet"
6199 type="htt:tTaskQueryResultSet"/>
6200             </xsd:sequence>
6201         </xsd:complexType>
6202     </xsd:element>
6203
6204     <xsd:element name="activate">
6205         <xsd:complexType>
6206             <xsd:sequence>
6207                 <xsd:element name="identifier" type="xsd:anyURI"/>
6208             </xsd:sequence>
6209         </xsd:complexType>
6210     </xsd:element>
6211     <xsd:element name="activateResponse">
6212         <xsd:complexType>
6213             <xsd:sequence>
6214                 <xsd:annotation>
6215                     <xsd:documentation>Empty message</xsd:documentation>
6216                 </xsd:annotation>
6217             </xsd:sequence>
6218         </xsd:complexType>
6219     </xsd:element>
6220
6221     <xsd:element name="batchActivate">
6222         <xsd:complexType>
6223             <xsd:sequence>
6224                 <xsd:element name="identifier" type="xsd:anyURI"
6225 maxOccurs="unbounded"/>
6226             </xsd:sequence>
6227         </xsd:complexType>
6228     </xsd:element>
6229     <xsd:element name="batchActivateResponse">
6230         <xsd:complexType>
6231             <xsd:sequence>
6232                 <xsd:element name="batchResponse" type="tBatchResponse"
6233 minOccurs="0" maxOccurs="unbounded"/>
6234             </xsd:sequence>
6235         </xsd:complexType>

```

```

6236     </xsd:element>
6237
6238     <xsd:element name="nominate">
6239         <xsd:complexType>
6240             <xsd:sequence>
6241                 <xsd:element name="identifier" type="xsd:anyURI"/>
6242                 <xsd:element name="organizationalEntity"
6243 type="htt:tOrganizationalEntity"/>
6244             </xsd:sequence>
6245         </xsd:complexType>
6246     </xsd:element>
6247     <xsd:element name="nominateResponse">
6248         <xsd:complexType>
6249             <xsd:sequence>
6250                 <xsd:annotation>
6251                     <xsd:documentation>Empty message</xsd:documentation>
6252                 </xsd:annotation>
6253             </xsd:sequence>
6254         </xsd:complexType>
6255     </xsd:element>
6256
6257     <xsd:element name="batchNominate">
6258         <xsd:complexType>
6259             <xsd:sequence>
6260                 <xsd:element name="identifier" type="xsd:anyURI"
6261 maxOccurs="unbounded"/>
6262             </xsd:sequence>
6263         </xsd:complexType>
6264     </xsd:element>
6265     <xsd:element name="batchNominateResponse">
6266         <xsd:complexType>
6267             <xsd:sequence>
6268                 <xsd:element name="batchResponse" type="tBatchResponse"
6269 minOccurs="0" maxOccurs="unbounded"/>
6270             </xsd:sequence>
6271         </xsd:complexType>
6272     </xsd:element>
6273
6274     <xsd:element name="setGenericHumanRole">
6275         <xsd:complexType>
6276             <xsd:sequence>
6277                 <xsd:element name="identifier" type="xsd:anyURI"/>
6278                 <xsd:element name="genericHumanRole" type="xsd:string"/>
6279                 <xsd:element name="organizationalEntity"
6280 type="htt:tOrganizationalEntity"/>
6281             </xsd:sequence>
6282         </xsd:complexType>
6283     </xsd:element>
6284     <xsd:element name="setGenericHumanRoleResponse">
6285         <xsd:complexType>
6286             <xsd:sequence>
6287                 <xsd:annotation>
6288                     <xsd:documentation>Empty message</xsd:documentation>
6289                 </xsd:annotation>
6290             </xsd:sequence>
6291         </xsd:complexType>
6292     </xsd:element>
6293

```

```

6294     <xsd:element name="batchSetGenericHumanRole">
6295         <xsd:complexType>
6296             <xsd:sequence>
6297                 <xsd:element name="identifier" type="xsd:anyURI"
6298 maxOccurs="unbounded"/>
6299                 <xsd:element name="genericHumanRole" type="xsd:string"/>
6300                 <xsd:element name="organizationalEntity"
6301 type="htt:tOrganizationalEntity"/>
6302             </xsd:sequence>
6303         </xsd:complexType>
6304     </xsd:element>
6305     <xsd:element name="batchSetGenericHumanRoleResponse">
6306         <xsd:complexType>
6307             <xsd:sequence>
6308                 <xsd:element name="batchResponse" type="tBatchResponse"
6309 minOccurs="0" maxOccurs="unbounded"/>
6310             </xsd:sequence>
6311         </xsd:complexType>
6312     </xsd:element>
6313
6314     <!-- Fault elements -->
6315     <xsd:element name="illegalState">
6316         <xsd:complexType>
6317             <xsd:sequence>
6318                 <xsd:element name="status" type="htt:tStatus"/>
6319                 <xsd:element name="message" type="xsd:string"/>
6320             </xsd:sequence>
6321         </xsd:complexType>
6322     </xsd:element>
6323
6324     <xsd:element name="illegalArgument" type="xsd:string"/>
6325
6326     <xsd:element name="illegalAccess" type="xsd:string"/>
6327
6328     <xsd:element name="illegalOperation" type="xsd:string"/>
6329
6330     <xsd:element name="recipientNotAllowed" type="xsd:string"/>
6331
6332     <xsd:complexType name="tBatchResponse">
6333         <xsd:sequence>
6334             <xsd:element name="identifier" type="xsd:anyURI"/>
6335             <xsd:choice>
6336                 <xsd:element ref="illegalState"/>
6337                 <xsd:element ref="illegalArgument"/>
6338                 <xsd:element ref="illegalAccess"/>
6339                 <xsd:element ref="illegalOperation"/>
6340                 <xsd:element ref="recipientNotAllowed"/>
6341                 <xsd:any namespace="##other" processContents="lax"/>
6342             </xsd:choice>
6343         </xsd:sequence>
6344     </xsd:complexType>
6345
6346 </xsd:schema>
6347 </wsdl:types>
6348
6349 <!-- Declaration of messages -->
6350 <wsdl:message name="addAttachment">
6351     <wsdl:part name="addAttachment" element="addAttachment"/>

```

```

6352 </wsdl:message>
6353 <wsdl:message name="addAttachmentResponse">
6354   <wsdl:part name="addAttachmentResponse" element="addAttachmentResponse"/>
6355 </wsdl:message>
6356
6357 <wsdl:message name="addComment">
6358   <wsdl:part name="addComment" element="addComment"/>
6359 </wsdl:message>
6360 <wsdl:message name="addCommentResponse">
6361   <wsdl:part name="addCommentResponse" element="addCommentResponse"/>
6362 </wsdl:message>
6363
6364 <wsdl:message name="claim">
6365   <wsdl:part name="claim" element="claim"/>
6366 </wsdl:message>
6367 <wsdl:message name="claimResponse">
6368   <wsdl:part name="claimResponse" element="claimResponse"/>
6369 </wsdl:message>
6370
6371 <wsdl:message name="batchClaim">
6372   <wsdl:part name="batchClaim" element="batchClaim"/>
6373 </wsdl:message>
6374 <wsdl:message name="batchClaimResponse">
6375   <wsdl:part name="batchClaimResponse" element="batchClaimResponse"/>
6376 </wsdl:message>
6377
6378 <wsdl:message name="complete">
6379   <wsdl:part name="complete" element="complete"/>
6380 </wsdl:message>
6381 <wsdl:message name="completeResponse">
6382   <wsdl:part name="completeResponse" element="completeResponse"/>
6383 </wsdl:message>
6384
6385 <wsdl:message name="batchComplete">
6386   <wsdl:part name="batchComplete" element="batchComplete"/>
6387 </wsdl:message>
6388 <wsdl:message name="batchCompleteResponse">
6389   <wsdl:part name="batchCompleteResponse" element="batchCompleteResponse"/>
6390 </wsdl:message>
6391
6392 <wsdl:message name="delegate">
6393   <wsdl:part name="delegate" element="delegate"/>
6394 </wsdl:message>
6395 <wsdl:message name="delegateResponse">
6396   <wsdl:part name="delegateResponse" element="delegateResponse"/>
6397 </wsdl:message>
6398
6399 <wsdl:message name="batchDelegate">
6400   <wsdl:part name="batchDelegate" element="batchDelegate"/>
6401 </wsdl:message>
6402 <wsdl:message name="batchDelegateResponse">
6403   <wsdl:part name="batchDelegateResponse" element="batchDelegateResponse"/>
6404 </wsdl:message>
6405
6406 <wsdl:message name="deleteAttachment">
6407   <wsdl:part name="deleteAttachment" element="deleteAttachment"/>
6408 </wsdl:message>
6409 <wsdl:message name="deleteAttachmentResponse">

```

```

6410     <wsdl:part name="deleteAttachmentResponse"
6411 element="deleteAttachmentResponse"/>
6412 </wsdl:message>
6413
6414 <wsdl:message name="deleteComment">
6415     <wsdl:part name="deleteComment" element="deleteComment"/>
6416 </wsdl:message>
6417 <wsdl:message name="deleteCommentResponse">
6418     <wsdl:part name="deleteCommentResponse" element="deleteCommentResponse"/>
6419 </wsdl:message>
6420
6421 <wsdl:message name="deleteFault">
6422     <wsdl:part name="deleteFault" element="deleteFault"/>
6423 </wsdl:message>
6424 <wsdl:message name="deleteFaultResponse">
6425     <wsdl:part name="deleteFaultResponse" element="deleteFaultResponse"/>
6426 </wsdl:message>
6427
6428 <wsdl:message name="deleteOutput">
6429     <wsdl:part name="deleteOutput" element="deleteOutput"/>
6430 </wsdl:message>
6431 <wsdl:message name="deleteOutputResponse">
6432     <wsdl:part name="deleteOutputResponse" element="deleteOutputResponse"/>
6433 </wsdl:message>
6434
6435 <wsdl:message name="fail">
6436     <wsdl:part name="fail" element="fail"/>
6437 </wsdl:message>
6438 <wsdl:message name="failResponse">
6439     <wsdl:part name="failResponse" element="failResponse"/>
6440 </wsdl:message>
6441
6442 <wsdl:message name="batchFail">
6443     <wsdl:part name="batchFail" element="batchFail"/>
6444 </wsdl:message>
6445 <wsdl:message name="batchFailResponse">
6446     <wsdl:part name="batchFailResponse" element="batchFailResponse"/>
6447 </wsdl:message>
6448
6449 <wsdl:message name="forward">
6450     <wsdl:part name="forward" element="forward"/>
6451 </wsdl:message>
6452 <wsdl:message name="forwardResponse">
6453     <wsdl:part name="forwardResponse" element="forwardResponse"/>
6454 </wsdl:message>
6455
6456 <wsdl:message name="batchForward">
6457     <wsdl:part name="batchForward" element="batchForward"/>
6458 </wsdl:message>
6459 <wsdl:message name="batchForwardResponse">
6460     <wsdl:part name="batchForwardResponse" element="batchForwardResponse"/>
6461 </wsdl:message>
6462
6463 <wsdl:message name="getAttachment">
6464     <wsdl:part name="getAttachment" element="getAttachment"/>
6465 </wsdl:message>
6466 <wsdl:message name="getAttachmentResponse">
6467     <wsdl:part name="getAttachmentResponse" element="getAttachmentResponse"/>

```

```

6468 </wsdl:message>
6469
6470 <wsdl:message name="getAttachmentInfos">
6471   <wsdl:part name="getAttachmentInfos" element="getAttachmentInfos"/>
6472 </wsdl:message>
6473 <wsdl:message name="getAttachmentInfosResponse">
6474   <wsdl:part name="getAttachmentInfosResponse"
6475 element="getAttachmentInfosResponse"/>
6476 </wsdl:message>
6477
6478 <wsdl:message name="getComments">
6479   <wsdl:part name="getComments" element="getComments"/>
6480 </wsdl:message>
6481 <wsdl:message name="getCommentsResponse">
6482   <wsdl:part name="getCommentsResponse" element="getCommentsResponse"/>
6483 </wsdl:message>
6484
6485 <wsdl:message name="getFault">
6486   <wsdl:part name="getFault" element="getFault"/>
6487 </wsdl:message>
6488 <wsdl:message name="getFaultResponse">
6489   <wsdl:part name="getFaultResponse" element="getFaultResponse"/>
6490 </wsdl:message>
6491
6492 <wsdl:message name="getInput">
6493   <wsdl:part name="getInput" element="getInput"/>
6494 </wsdl:message>
6495 <wsdl:message name="getInputResponse">
6496   <wsdl:part name="getInputResponse" element="getInputResponse"/>
6497 </wsdl:message>
6498
6499 <wsdl:message name="getOutcome">
6500   <wsdl:part name="getOutcome" element="getOutcome"/>
6501 </wsdl:message>
6502 <wsdl:message name="getOutcomeResponse">
6503   <wsdl:part name="getOutcomeResponse" element="getOutcomeResponse"/>
6504 </wsdl:message>
6505
6506 <wsdl:message name="getOutput">
6507   <wsdl:part name="getOutput" element="getOutput"/>
6508 </wsdl:message>
6509 <wsdl:message name="getOutputResponse">
6510   <wsdl:part name="getOutputResponse" element="getOutputResponse"/>
6511 </wsdl:message>
6512
6513 <wsdl:message name="getParentTask">
6514   <wsdl:part name="getParentTask" element="getParentTask"/>
6515 </wsdl:message>
6516 <wsdl:message name="getParentTaskResponse">
6517   <wsdl:part name="getParentTaskResponse" element="getParentTaskResponse"/>
6518 </wsdl:message>
6519
6520 <wsdl:message name="getParentTaskIdentifier">
6521   <wsdl:part name="getParentTaskIdentifier"
6522 element="getParentTaskIdentifier"/>
6523 </wsdl:message>
6524 <wsdl:message name="getParentTaskIdentifierResponse">

```

```

6525     <wsdl:part name="getParentTaskIdentifierResponse"
6526 element="getParentTaskIdentifierResponse"/>
6527 </wsdl:message>
6528
6529 <wsdl:message name="getRendering">
6530 <wsdl:part name="getRendering" element="getRendering"/>
6531 </wsdl:message>
6532 <wsdl:message name="getRenderingResponse">
6533 <wsdl:part name="getRenderingResponse" element="getRenderingResponse"/>
6534 </wsdl:message>
6535
6536 <wsdl:message name="getRenderingTypes">
6537 <wsdl:part name="getRenderingTypes" element="getRenderingTypes"/>
6538 </wsdl:message>
6539 <wsdl:message name="getRenderingTypesResponse">
6540 <wsdl:part name="getRenderingTypesResponse"
6541 element="getRenderingTypesResponse"/>
6542 </wsdl:message>
6543
6544 <wsdl:message name="getSubtaskIdentifiers">
6545 <wsdl:part name="getSubtaskIdentifiers" element="getSubtaskIdentifiers"/>
6546 </wsdl:message>
6547 <wsdl:message name="getSubtaskIdentifiersResponse">
6548 <wsdl:part name="getSubtaskIdentifiersResponse"
6549 element="getSubtaskIdentifiersResponse"/>
6550 </wsdl:message>
6551
6552 <wsdl:message name="getSubtasks">
6553 <wsdl:part name="getSubtasks" element="getSubtasks"/>
6554 </wsdl:message>
6555 <wsdl:message name="getSubtasksResponse">
6556 <wsdl:part name="getSubtasksResponse" element="getSubtasksResponse"/>
6557 </wsdl:message>
6558
6559 <wsdl:message name="getTaskDescription">
6560 <wsdl:part name="getTaskDescription" element="getTaskDescription"/>
6561 </wsdl:message>
6562 <wsdl:message name="getTaskDescriptionResponse">
6563 <wsdl:part name="getTaskDescriptionResponse"
6564 element="getTaskDescriptionResponse"/>
6565 </wsdl:message>
6566
6567 <wsdl:message name="getTaskDetails">
6568 <wsdl:part name="getTaskDetails" element="getTaskDetails"/>
6569 </wsdl:message>
6570 <wsdl:message name="getTaskDetailsResponse">
6571 <wsdl:part name="getTaskDetailsResponse"
6572 element="getTaskDetailsResponse"/>
6573 </wsdl:message>
6574
6575 <wsdl:message name="getTaskHistory">
6576 <wsdl:part name="getTaskHistory" element="getTaskHistory"/>
6577 </wsdl:message>
6578 <wsdl:message name="getTaskHistoryResponse">
6579 <wsdl:part name="getTaskHistoryResponse"
6580 element="getTaskHistoryResponse"/>
6581 </wsdl:message>
6582

```

```

6583 <wsdl:message name="getTaskInstanceData">
6584   <wsdl:part name="getTaskInstanceData" element="getTaskInstanceData"/>
6585 </wsdl:message>
6586 <wsdl:message name="getTaskInstanceDataResponse">
6587   <wsdl:part name="getTaskInstanceDataResponse"
6588 element="getTaskInstanceDataResponse"/>
6589 </wsdl:message>
6590
6591 <wsdl:message name="getTaskOperations">
6592   <wsdl:part name="getTaskOperations" element="getTaskOperations"/>
6593 </wsdl:message>
6594 <wsdl:message name="getTaskOperationsResponse">
6595   <wsdl:part name="getTaskOperationsResponse"
6596 element="getTaskOperationsResponse"/>
6597 </wsdl:message>
6598
6599 <wsdl:message name="hasSubtasks">
6600   <wsdl:part name="hasSubtasks" element="hasSubtasks"/>
6601 </wsdl:message>
6602 <wsdl:message name="hasSubtasksResponse">
6603   <wsdl:part name="hasSubtasksResponse" element="hasSubtasksResponse"/>
6604 </wsdl:message>
6605
6606 <wsdl:message name="instantiateSubtask">
6607   <wsdl:part name="instantiateSubtask" element="instantiateSubtask"/>
6608 </wsdl:message>
6609 <wsdl:message name="instantiateSubtaskResponse">
6610   <wsdl:part name="instantiateSubtaskResponse"
6611 element="instantiateSubtaskResponse"/>
6612 </wsdl:message>
6613
6614 <wsdl:message name="isSubtask">
6615   <wsdl:part name="isSubtask" element="isSubtask"/>
6616 </wsdl:message>
6617 <wsdl:message name="isSubtaskResponse">
6618   <wsdl:part name="isSubtaskResponse" element="isSubtaskResponse"/>
6619 </wsdl:message>
6620
6621 <wsdl:message name="release">
6622   <wsdl:part name="release" element="release"/>
6623 </wsdl:message>
6624 <wsdl:message name="releaseResponse">
6625   <wsdl:part name="releaseResponse" element="releaseResponse"/>
6626 </wsdl:message>
6627
6628 <wsdl:message name="batchRelease">
6629   <wsdl:part name="batchRelease" element="batchRelease"/>
6630 </wsdl:message>
6631 <wsdl:message name="batchReleaseResponse">
6632   <wsdl:part name="batchReleaseResponse" element="batchReleaseResponse"/>
6633 </wsdl:message>
6634
6635 <wsdl:message name="remove">
6636   <wsdl:part name="remove" element="remove"/>
6637 </wsdl:message>
6638 <wsdl:message name="removeResponse">
6639   <wsdl:part name="removeResponse" element="removeResponse"/>
6640 </wsdl:message>

```

```

6641 <wsdl:message name="batchRemove">
6642   <wsdl:part name="batchRemove" element="batchRemove"/>
6643 </wsdl:message>
6644 <wsdl:message name="batchRemoveResponse">
6645   <wsdl:part name="batchRemoveResponse" element="batchRemoveResponse"/>
6646 </wsdl:message>
6647
6648
6649 <wsdl:message name="resume">
6650   <wsdl:part name="resume" element="resume"/>
6651 </wsdl:message>
6652 <wsdl:message name="resumeResponse">
6653   <wsdl:part name="resumeResponse" element="resumeResponse"/>
6654 </wsdl:message>
6655
6656 <wsdl:message name="batchResume">
6657   <wsdl:part name="batchResume" element="batchResume"/>
6658 </wsdl:message>
6659 <wsdl:message name="batchResumeResponse">
6660   <wsdl:part name="batchResumeResponse" element="batchResumeResponse"/>
6661 </wsdl:message>
6662
6663 <wsdl:message name="setFault">
6664   <wsdl:part name="setFault" element="setFault"/>
6665 </wsdl:message>
6666 <wsdl:message name="setFaultResponse">
6667   <wsdl:part name="setFaultResponse" element="setFaultResponse"/>
6668 </wsdl:message>
6669
6670 <wsdl:message name="setOutput">
6671   <wsdl:part name="setOutput" element="setOutput"/>
6672 </wsdl:message>
6673 <wsdl:message name="setOutputResponse">
6674   <wsdl:part name="setOutputResponse" element="setOutputResponse"/>
6675 </wsdl:message>
6676
6677 <wsdl:message name="setPriority">
6678   <wsdl:part name="setPriority" element="setPriority"/>
6679 </wsdl:message>
6680 <wsdl:message name="setPriorityResponse">
6681   <wsdl:part name="setPriorityResponse" element="setPriorityResponse"/>
6682 </wsdl:message>
6683
6684 <wsdl:message name="batchSetPriority">
6685   <wsdl:part name="batchSetPriority" element="batchSetPriority"/>
6686 </wsdl:message>
6687 <wsdl:message name="batchSetPriorityResponse">
6688   <wsdl:part name="batchSetPriorityResponse"
6689 element="batchSetPriorityResponse"/>
6690 </wsdl:message>
6691
6692 <wsdl:message name="setTaskCompletionDeadlineExpression">
6693   <wsdl:part name="setTaskCompletionDeadlineExpression"
6694 element="setTaskCompletionDeadlineExpression"/>
6695 </wsdl:message>
6696 <wsdl:message name="setTaskCompletionDeadlineExpressionResponse">
6697   <wsdl:part name="setTaskCompletionDeadlineExpressionResponse"
6698 element="setTaskCompletionDeadlineExpressionResponse"/>

```

```

6699 </wsdl:message>
6700
6701 <wsdl:message name="setTaskCompletionDurationExpression">
6702 <wsdl:part name="setTaskCompletionDurationExpression"
6703 element="setTaskCompletionDurationExpression"/>
6704 </wsdl:message>
6705 <wsdl:message name="setTaskCompletionDurationExpressionResponse">
6706 <wsdl:part name="setTaskCompletionDurationExpressionResponse"
6707 element="setTaskCompletionDurationExpressionResponse"/>
6708 </wsdl:message>
6709
6710 <wsdl:message name="setTaskStartDeadlineExpression">
6711 <wsdl:part name="setTaskStartDeadlineExpression"
6712 element="setTaskStartDeadlineExpression"/>
6713 </wsdl:message>
6714 <wsdl:message name="setTaskStartDeadlineExpressionResponse">
6715 <wsdl:part name="setTaskStartDeadlineExpressionResponse"
6716 element="setTaskStartDeadlineExpressionResponse"/>
6717 </wsdl:message>
6718
6719 <wsdl:message name="setTaskStartDurationExpression">
6720 <wsdl:part name="setTaskStartDurationExpression"
6721 element="setTaskStartDurationExpression"/>
6722 </wsdl:message>
6723 <wsdl:message name="setTaskStartDurationExpressionResponse">
6724 <wsdl:part name="setTaskStartDurationExpressionResponse"
6725 element="setTaskStartDurationExpressionResponse"/>
6726 </wsdl:message>
6727
6728 <wsdl:message name="skip">
6729 <wsdl:part name="skip" element="skip"/>
6730 </wsdl:message>
6731 <wsdl:message name="skipResponse">
6732 <wsdl:part name="skipResponse" element="skipResponse"/>
6733 </wsdl:message>
6734
6735 <wsdl:message name="batchSkip">
6736 <wsdl:part name="batchSkip" element="batchSkip"/>
6737 </wsdl:message>
6738 <wsdl:message name="batchSkipResponse">
6739 <wsdl:part name="batchSkipResponse" element="batchSkipResponse"/>
6740 </wsdl:message>
6741
6742 <wsdl:message name="start">
6743 <wsdl:part name="start" element="start"/>
6744 </wsdl:message>
6745 <wsdl:message name="startResponse">
6746 <wsdl:part name="startResponse" element="startResponse"/>
6747 </wsdl:message>
6748
6749 <wsdl:message name="batchStart">
6750 <wsdl:part name="batchStart" element="batchStart"/>
6751 </wsdl:message>
6752 <wsdl:message name="batchStartResponse">
6753 <wsdl:part name="batchStartResponse" element="batchStartResponse"/>
6754 </wsdl:message>
6755
6756 <wsdl:message name="stop">

```

```

6757     <wsdl:part name="stop" element="stop"/>
6758 </wsdl:message>
6759 <wsdl:message name="stopResponse">
6760     <wsdl:part name="stopResponse" element="stopResponse"/>
6761 </wsdl:message>
6762
6763 <wsdl:message name="batchStop">
6764     <wsdl:part name="batchStop" element="batchStop"/>
6765 </wsdl:message>
6766 <wsdl:message name="batchStopResponse">
6767     <wsdl:part name="batchStopResponse" element="batchStopResponse"/>
6768 </wsdl:message>
6769
6770 <wsdl:message name="suspend">
6771     <wsdl:part name="suspend" element="suspend"/>
6772 </wsdl:message>
6773 <wsdl:message name="suspendResponse">
6774     <wsdl:part name="suspendResponse" element="suspendResponse"/>
6775 </wsdl:message>
6776
6777 <wsdl:message name="batchSuspend">
6778     <wsdl:part name="batchSuspend" element="batchSuspend"/>
6779 </wsdl:message>
6780 <wsdl:message name="batchSuspendResponse">
6781     <wsdl:part name="batchSuspendResponse" element="batchSuspendResponse"/>
6782 </wsdl:message>
6783
6784 <wsdl:message name="suspendUntil">
6785     <wsdl:part name="suspendUntil" element="suspendUntil"/>
6786 </wsdl:message>
6787 <wsdl:message name="suspendUntilResponse">
6788     <wsdl:part name="suspendUntilResponse" element="suspendUntilResponse"/>
6789 </wsdl:message>
6790
6791 <wsdl:message name="batchSuspendUntil">
6792     <wsdl:part name="batchSuspendUntil" element="batchSuspendUntil"/>
6793 </wsdl:message>
6794 <wsdl:message name="batchSuspendUntilResponse">
6795     <wsdl:part name="batchSuspendUntilResponse"
6796 element="batchSuspendUntilResponse"/>
6797 </wsdl:message>
6798
6799 <wsdl:message name="updateComment">
6800     <wsdl:part name="updateComment" element="updateComment"/>
6801 </wsdl:message>
6802 <wsdl:message name="updateCommentResponse">
6803     <wsdl:part name="updateCommentResponse" element="updateCommentResponse"/>
6804 </wsdl:message>
6805
6806 <wsdl:message name="getMyTaskAbstracts">
6807     <wsdl:part name="getMyTaskAbstracts" element="getMyTaskAbstracts"/>
6808 </wsdl:message>
6809 <wsdl:message name="getMyTaskAbstractsResponse">
6810     <wsdl:part name="getMyTaskAbstractsResponse"
6811 element="getMyTaskAbstractsResponse"/>
6812 </wsdl:message>
6813
6814 <wsdl:message name="getMyTaskDetails">

```

```

6815     <wsdl:part name="getMyTaskDetails" element="getMyTaskDetails"/>
6816 </wsdl:message>
6817 <wsdl:message name="getMyTaskDetailsResponse">
6818     <wsdl:part name="getMyTaskDetailsResponse"
6819 element="getMyTaskDetailsResponse"/>
6820 </wsdl:message>
6821
6822 <wsdl:message name="query">
6823     <wsdl:part name="query" element="query"/>
6824 </wsdl:message>
6825 <wsdl:message name="queryResponse">
6826     <wsdl:part name="queryResponse" element="queryResponse"/>
6827 </wsdl:message>
6828
6829 <wsdl:message name="activate">
6830     <wsdl:part name="activate" element="activate"/>
6831 </wsdl:message>
6832 <wsdl:message name="activateResponse">
6833     <wsdl:part name="activateResponse" element="activateResponse"/>
6834 </wsdl:message>
6835
6836 <wsdl:message name="batchActivate">
6837     <wsdl:part name="batchActivate" element="batchActivate"/>
6838 </wsdl:message>
6839 <wsdl:message name="batchActivateResponse">
6840     <wsdl:part name="batchActivateResponse" element="batchActivateResponse"/>
6841 </wsdl:message>
6842
6843 <wsdl:message name="nominate">
6844     <wsdl:part name="nominate" element="nominate"/>
6845 </wsdl:message>
6846 <wsdl:message name="nominateResponse">
6847     <wsdl:part name="nominateResponse" element="nominateResponse"/>
6848 </wsdl:message>
6849
6850 <wsdl:message name="batchNominate">
6851     <wsdl:part name="batchNominate" element="batchNominate"/>
6852 </wsdl:message>
6853 <wsdl:message name="batchNominateResponse">
6854     <wsdl:part name="batchNominateResponse" element="batchNominateResponse"/>
6855 </wsdl:message>
6856
6857 <wsdl:message name="setGenericHumanRole">
6858     <wsdl:part name="setGenericHumanRole" element="setGenericHumanRole"/>
6859 </wsdl:message>
6860 <wsdl:message name="setGenericHumanRoleResponse">
6861     <wsdl:part name="setGenericHumanRoleResponse"
6862 element="setGenericHumanRoleResponse"/>
6863 </wsdl:message>
6864
6865 <wsdl:message name="batchSetGenericHumanRole">
6866     <wsdl:part name="batchSetGenericHumanRole"
6867 element="batchSetGenericHumanRole"/>
6868 </wsdl:message>
6869 <wsdl:message name="batchSetGenericHumanRoleResponse">
6870     <wsdl:part name="batchSetGenericHumanRoleResponse"
6871 element="batchSetGenericHumanRoleResponse"/>
6872 </wsdl:message>

```

```

6873
6874 <!-- Declaration of fault messages -->
6875 <wsdl:message name="illegalStateFault">
6876   <wsdl:part name="illegalState" element="illegalState"/>
6877 </wsdl:message>
6878 <wsdl:message name="illegalArgumentFault">
6879   <wsdl:part name="illegalArgument" element="illegalArgument"/>
6880 </wsdl:message>
6881 <wsdl:message name="illegalAccessFault">
6882   <wsdl:part name="illegalAccess" element="illegalAccess"/>
6883 </wsdl:message>
6884 <wsdl:message name="illegalOperationFault">
6885   <wsdl:part name="illegalOperation" element="illegalOperation"/>
6886 </wsdl:message>
6887 <wsdl:message name="recipientNotAllowed">
6888   <wsdl:part name="recipientNotAllowed" element="recipientNotAllowed"/>
6889 </wsdl:message>
6890
6891 <!-- Port type definition -->
6892 <wsdl:portType name="taskOperations">
6893
6894   <wsdl:operation name="addAttachment">
6895     <wsdl:input message="addAttachment"/>
6896     <wsdl:output message="addAttachmentResponse"/>
6897     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
6898     <wsdl:fault name="illegalArgumentFault"
6899 message="illegalArgumentFault"/>
6900     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
6901     <wsdl:fault name="illegalOperationFault"
6902 message="illegalOperationFault"/>
6903   </wsdl:operation>
6904
6905   <wsdl:operation name="addComment">
6906     <wsdl:input message="addComment"/>
6907     <wsdl:output message="addCommentResponse"/>
6908     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
6909     <wsdl:fault name="illegalArgumentFault"
6910 message="illegalArgumentFault"/>
6911     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
6912     <wsdl:fault name="illegalOperationFault"
6913 message="illegalOperationFault"/>
6914   </wsdl:operation>
6915
6916   <wsdl:operation name="claim">
6917     <wsdl:input message="claim"/>
6918     <wsdl:output message="claimResponse"/>
6919     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
6920     <wsdl:fault name="illegalArgumentFault"
6921 message="illegalArgumentFault"/>
6922     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
6923     <wsdl:fault name="illegalOperationFault"
6924 message="illegalOperationFault"/>
6925   </wsdl:operation>
6926
6927   <wsdl:operation name="batchClaim">
6928     <wsdl:input message="batchClaim"/>
6929     <wsdl:output message="batchClaimResponse"/>
6930   </wsdl:operation>

```

```

6931
6932     <wsdl:operation name="complete">
6933         <wsdl:input message="complete"/>
6934         <wsdl:output message="completeResponse"/>
6935         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
6936         <wsdl:fault name="illegalArgumentFault"
6937 message="illegalArgumentFault"/>
6938         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
6939         <wsdl:fault name="illegalOperationFault"
6940 message="illegalOperationFault"/>
6941     </wsdl:operation>
6942
6943     <wsdl:operation name="batchComplete">
6944         <wsdl:input message="batchComplete"/>
6945         <wsdl:output message="batchCompleteResponse"/>
6946     </wsdl:operation>
6947
6948     <wsdl:operation name="delegate">
6949         <wsdl:input message="delegate"/>
6950         <wsdl:output message="delegateResponse"/>
6951         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
6952         <wsdl:fault name="illegalArgumentFault"
6953 message="illegalArgumentFault"/>
6954         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
6955         <wsdl:fault name="illegalOperationFault"
6956 message="illegalOperationFault"/>
6957         <wsdl:fault name="recipientNotAllowed" message="recipientNotAllowed"/>
6958     </wsdl:operation>
6959
6960     <wsdl:operation name="batchDelegate">
6961         <wsdl:input message="batchDelegate"/>
6962         <wsdl:output message="batchDelegateResponse"/>
6963     </wsdl:operation>
6964
6965     <wsdl:operation name="deleteAttachment">
6966         <wsdl:input message="deleteAttachment"/>
6967         <wsdl:output message="deleteAttachmentResponse"/>
6968         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
6969         <wsdl:fault name="illegalArgumentFault"
6970 message="illegalArgumentFault"/>
6971         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
6972         <wsdl:fault name="illegalOperationFault"
6973 message="illegalOperationFault"/>
6974     </wsdl:operation>
6975
6976     <wsdl:operation name="deleteComment">
6977         <wsdl:input message="deleteComment"/>
6978         <wsdl:output message="deleteCommentResponse"/>
6979         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
6980         <wsdl:fault name="illegalArgumentFault"
6981 message="illegalArgumentFault"/>
6982         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
6983         <wsdl:fault name="illegalOperationFault"
6984 message="illegalOperationFault"/>
6985     </wsdl:operation>
6986
6987     <wsdl:operation name="deleteFault">
6988         <wsdl:input message="deleteFault"/>

```

```

6989     <wsdl:output message="deleteFaultResponse"/>
6990     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
6991     <wsdl:fault name="illegalArgumentFault"
6992 message="illegalArgumentFault"/>
6993     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
6994     <wsdl:fault name="illegalOperationFault"
6995 message="illegalOperationFault"/>
6996   </wsdl:operation>
6997
6998   <wsdl:operation name="deleteOutput">
6999     <wsdl:input message="deleteOutput"/>
7000     <wsdl:output message="deleteOutputResponse"/>
7001     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7002     <wsdl:fault name="illegalArgumentFault"
7003 message="illegalArgumentFault"/>
7004     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7005     <wsdl:fault name="illegalOperationFault"
7006 message="illegalOperationFault"/>
7007   </wsdl:operation>
7008
7009   <wsdl:operation name="fail">
7010     <wsdl:input message="fail"/>
7011     <wsdl:output message="failResponse"/>
7012     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7013     <wsdl:fault name="illegalArgumentFault"
7014 message="illegalArgumentFault"/>
7015     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7016     <wsdl:fault name="illegalOperationFault"
7017 message="illegalOperationFault"/>
7018   </wsdl:operation>
7019
7020   <wsdl:operation name="batchFail">
7021     <wsdl:input message="batchFail"/>
7022     <wsdl:output message="batchFailResponse"/>
7023   </wsdl:operation>
7024
7025   <wsdl:operation name="forward">
7026     <wsdl:input message="forward"/>
7027     <wsdl:output message="forwardResponse"/>
7028     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7029     <wsdl:fault name="illegalArgumentFault"
7030 message="illegalArgumentFault"/>
7031     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7032     <wsdl:fault name="illegalOperationFault"
7033 message="illegalOperationFault"/>
7034   </wsdl:operation>
7035
7036   <wsdl:operation name="batchForward">
7037     <wsdl:input message="batchForward"/>
7038     <wsdl:output message="batchForwardResponse"/>
7039   </wsdl:operation>
7040
7041   <wsdl:operation name="getAttachment">
7042     <wsdl:input message="getAttachment"/>
7043     <wsdl:output message="getAttachmentResponse"/>
7044     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7045     <wsdl:fault name="illegalArgumentFault"
7046 message="illegalArgumentFault"/>

```

```

7047     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7048     <wsdl:fault name="illegalOperationFault"
7049 message="illegalOperationFault"/>
7050 </wsdl:operation>
7051
7052     <wsdl:operation name="getAttachmentInfos">
7053     <wsdl:input message="getAttachmentInfos"/>
7054     <wsdl:output message="getAttachmentInfosResponse"/>
7055     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7056     <wsdl:fault name="illegalArgumentFault"
7057 message="illegalArgumentFault"/>
7058     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7059     <wsdl:fault name="illegalOperationFault"
7060 message="illegalOperationFault"/>
7061 </wsdl:operation>
7062
7063     <wsdl:operation name="getComments">
7064     <wsdl:input message="getComments"/>
7065     <wsdl:output message="getCommentsResponse"/>
7066     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7067     <wsdl:fault name="illegalArgumentFault"
7068 message="illegalArgumentFault"/>
7069     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7070     <wsdl:fault name="illegalOperationFault"
7071 message="illegalOperationFault"/>
7072 </wsdl:operation>
7073
7074     <wsdl:operation name="getFault">
7075     <wsdl:input message="getFault"/>
7076     <wsdl:output message="getFaultResponse"/>
7077     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7078     <wsdl:fault name="illegalArgumentFault"
7079 message="illegalArgumentFault"/>
7080     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7081     <wsdl:fault name="illegalOperationFault"
7082 message="illegalOperationFault"/>
7083 </wsdl:operation>
7084
7085     <wsdl:operation name="getInput">
7086     <wsdl:input message="getInput"/>
7087     <wsdl:output message="getInputResponse"/>
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="getOutcome">
7097     <wsdl:input message="getOutcome"/>
7098     <wsdl:output message="getOutcomeResponse"/>
7099     <wsdl:fault name="illegalArgumentFault"
7100 message="illegalArgumentFault"/>
7101     <wsdl:fault name="illegalOperationFault"
7102 message="illegalOperationFault"/>
7103 </wsdl:operation>
7104

```

```

7105     <wsdl:operation name="getOutput">
7106         <wsdl:input message="getOutput"/>
7107         <wsdl:output message="getOutputResponse"/>
7108         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7109         <wsdl:fault name="illegalArgumentFault"
7110 message="illegalArgumentFault"/>
7111         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7112         <wsdl:fault name="illegalOperationFault"
7113 message="illegalOperationFault"/>
7114     </wsdl:operation>
7115
7116     <wsdl:operation name="getParentTask">
7117         <wsdl:input message="getParentTask"/>
7118         <wsdl:output message="getParentTaskResponse"/>
7119         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7120         <wsdl:fault name="illegalArgumentFault"
7121 message="illegalArgumentFault"/>
7122         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7123         <wsdl:fault name="illegalOperationFault"
7124 message="illegalOperationFault"/>
7125     </wsdl:operation>
7126
7127     <wsdl:operation name="getParentTaskIdentifier">
7128         <wsdl:input message="getParentTaskIdentifier"/>
7129         <wsdl:output message="getParentTaskIdentifierResponse"/>
7130         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7131         <wsdl:fault name="illegalArgumentFault"
7132 message="illegalArgumentFault"/>
7133         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7134         <wsdl:fault name="illegalOperationFault"
7135 message="illegalOperationFault"/>
7136     </wsdl:operation>
7137
7138     <wsdl:operation name="getRendering">
7139         <wsdl:input message="getRendering"/>
7140         <wsdl:output message="getRenderingResponse"/>
7141         <wsdl:fault name="illegalArgumentFault"
7142 message="illegalArgumentFault"/>
7143     </wsdl:operation>
7144
7145     <wsdl:operation name="getRenderingTypes">
7146         <wsdl:input message="getRenderingTypes"/>
7147         <wsdl:output message="getRenderingTypesResponse"/>
7148         <wsdl:fault name="illegalArgumentFault"
7149 message="illegalArgumentFault"/>
7150     </wsdl:operation>
7151
7152     <wsdl:operation name="getSubtaskIdentifiers">
7153         <wsdl:input message="getSubtaskIdentifiers"/>
7154         <wsdl:output message="getSubtaskIdentifiersResponse"/>
7155         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7156         <wsdl:fault name="illegalArgumentFault"
7157 message="illegalArgumentFault"/>
7158         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7159         <wsdl:fault name="illegalOperationFault"
7160 message="illegalOperationFault"/>
7161     </wsdl:operation>
7162

```

```

7163     <wsdl:operation name="getSubtasks">
7164         <wsdl:input message="getSubtasks"/>
7165         <wsdl:output message="getSubtasksResponse"/>
7166         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7167         <wsdl:fault name="illegalArgumentFault"
7168 message="illegalArgumentFault"/>
7169         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7170         <wsdl:fault name="illegalOperationFault"
7171 message="illegalOperationFault"/>
7172     </wsdl:operation>
7173
7174     <wsdl:operation name="getTaskDescription">
7175         <wsdl:input message="getTaskDescription"/>
7176         <wsdl:output message="getTaskDescriptionResponse"/>
7177         <wsdl:fault name="illegalArgumentFault"
7178 message="illegalArgumentFault"/>
7179     </wsdl:operation>
7180
7181     <wsdl:operation name="getTaskDetails">
7182         <wsdl:input message="getTaskDetails"/>
7183         <wsdl:output message="getTaskDetailsResponse"/>
7184         <wsdl:fault name="illegalArgumentFault"
7185 message="illegalArgumentFault"/>
7186     </wsdl:operation>
7187
7188     <wsdl:operation name="getTaskHistory">
7189         <wsdl:input message="getTaskHistory"/>
7190         <wsdl:output message="getTaskHistoryResponse"/>
7191         <wsdl:fault name="illegalArgumentFault"
7192 message="illegalArgumentFault"/>
7193         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7194         <wsdl:fault name="illegalOperationFault"
7195 message="illegalOperationFault"/>
7196     </wsdl:operation>
7197
7198     <wsdl:operation name="getTaskInstanceData">
7199         <wsdl:input message="getTaskInstanceData"/>
7200         <wsdl:output message="getTaskInstanceDataResponse"/>
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="getTaskOperations">
7209         <wsdl:input message="getTaskOperations"/>
7210         <wsdl:output message="getTaskOperationsResponse"/>
7211         <wsdl:fault name="illegalArgumentFault"
7212 message="illegalArgumentFault"/>
7213         <wsdl:fault name="illegalOperationFault"
7214 message="illegalOperationFault"/>
7215     </wsdl:operation>
7216
7217     <wsdl:operation name="hasSubtasks">
7218         <wsdl:input message="hasSubtasks"/>
7219         <wsdl:output message="hasSubtasksResponse"/>
7220         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>

```

```

7221     <wsdl:fault name="illegalArgumentFault"
7222 message="illegalArgumentFault"/>
7223     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7224     <wsdl:fault name="illegalOperationFault"
7225 message="illegalOperationFault"/>
7226     </wsdl:operation>
7227
7228     <wsdl:operation name="instantiateSubtask">
7229     <wsdl:input message="instantiateSubtask"/>
7230     <wsdl:output message="instantiateSubtaskResponse"/>
7231     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7232     <wsdl:fault name="illegalArgumentFault"
7233 message="illegalArgumentFault"/>
7234     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7235     <wsdl:fault name="illegalOperationFault"
7236 message="illegalOperationFault"/>
7237     </wsdl:operation>
7238
7239     <wsdl:operation name="isSubtask">
7240     <wsdl:input message="isSubtask"/>
7241     <wsdl:output message="isSubtaskResponse"/>
7242     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7243     <wsdl:fault name="illegalArgumentFault"
7244 message="illegalArgumentFault"/>
7245     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7246     <wsdl:fault name="illegalOperationFault"
7247 message="illegalOperationFault"/>
7248     </wsdl:operation>
7249
7250     <wsdl:operation name="release">
7251     <wsdl:input message="release"/>
7252     <wsdl:output message="releaseResponse"/>
7253     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7254     <wsdl:fault name="illegalArgumentFault"
7255 message="illegalArgumentFault"/>
7256     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7257     <wsdl:fault name="illegalOperationFault"
7258 message="illegalOperationFault"/>
7259     </wsdl:operation>
7260
7261     <wsdl:operation name="batchRelease">
7262     <wsdl:input message="batchRelease"/>
7263     <wsdl:output message="batchReleaseResponse"/>
7264     </wsdl:operation>
7265
7266     <wsdl:operation name="remove">
7267     <wsdl:input message="remove"/>
7268     <wsdl:output message="removeResponse"/>
7269     <wsdl:fault name="illegalArgumentFault"
7270 message="illegalArgumentFault"/>
7271     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7272     <wsdl:fault name="illegalOperationFault"
7273 message="illegalOperationFault"/>
7274     </wsdl:operation>
7275
7276     <wsdl:operation name="batchRemove">
7277     <wsdl:input message="batchRemove"/>
7278     <wsdl:output message="batchRemoveResponse"/>

```

```

7279     </wsdl:operation>
7280
7281     <wsdl:operation name="resume">
7282         <wsdl:input message="resume"/>
7283         <wsdl:output message="resumeResponse"/>
7284         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7285         <wsdl:fault name="illegalArgumentFault"
7286 message="illegalArgumentFault"/>
7287         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7288         <wsdl:fault name="illegalOperationFault"
7289 message="illegalOperationFault"/>
7290     </wsdl:operation>
7291
7292     <wsdl:operation name="batchResume">
7293         <wsdl:input message="batchResume"/>
7294         <wsdl:output message="batchResumeResponse"/>
7295     </wsdl:operation>
7296
7297     <wsdl:operation name="setFault">
7298         <wsdl:input message="setFault"/>
7299         <wsdl:output message="setFaultResponse"/>
7300         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7301         <wsdl:fault name="illegalArgumentFault"
7302 message="illegalArgumentFault"/>
7303         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7304         <wsdl:fault name="illegalOperationFault"
7305 message="illegalOperationFault"/>
7306     </wsdl:operation>
7307
7308     <wsdl:operation name="setOutput">
7309         <wsdl:input message="setOutput"/>
7310         <wsdl:output message="setOutputResponse"/>
7311         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7312         <wsdl:fault name="illegalArgumentFault"
7313 message="illegalArgumentFault"/>
7314         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7315         <wsdl:fault name="illegalOperationFault"
7316 message="illegalOperationFault"/>
7317     </wsdl:operation>
7318
7319     <wsdl:operation name="setPriority">
7320         <wsdl:input message="setPriority"/>
7321         <wsdl:output message="setPriorityResponse"/>
7322         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7323         <wsdl:fault name="illegalArgumentFault"
7324 message="illegalArgumentFault"/>
7325         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7326         <wsdl:fault name="illegalOperationFault"
7327 message="illegalOperationFault"/>
7328     </wsdl:operation>
7329
7330     <wsdl:operation name="batchSetPriority">
7331         <wsdl:input message="batchSetPriority"/>
7332         <wsdl:output message="batchSetPriorityResponse"/>
7333     </wsdl:operation>
7334
7335     <wsdl:operation name="setTaskCompletionDeadlineExpression">
7336         <wsdl:input message="setTaskCompletionDeadlineExpression"/>

```

```

7337     <wsdl:output message="setTaskCompletionDeadlineExpressionResponse"/>
7338     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7339     <wsdl:fault name="illegalArgumentFault"
7340 message="illegalArgumentFault"/>
7341     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7342     <wsdl:fault name="illegalOperationFault"
7343 message="illegalOperationFault"/>
7344     </wsdl:operation>
7345
7346     <wsdl:operation name="setTaskCompletionDurationExpression">
7347     <wsdl:input message="setTaskCompletionDurationExpression"/>
7348     <wsdl:output message="setTaskCompletionDurationExpressionResponse"/>
7349     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7350     <wsdl:fault name="illegalArgumentFault"
7351 message="illegalArgumentFault"/>
7352     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7353     <wsdl:fault name="illegalOperationFault"
7354 message="illegalOperationFault"/>
7355     </wsdl:operation>
7356
7357     <wsdl:operation name="setTaskStartDeadlineExpression">
7358     <wsdl:input message="setTaskStartDeadlineExpression"/>
7359     <wsdl:output message="setTaskStartDeadlineExpressionResponse"/>
7360     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7361     <wsdl:fault name="illegalArgumentFault"
7362 message="illegalArgumentFault"/>
7363     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7364     <wsdl:fault name="illegalOperationFault"
7365 message="illegalOperationFault"/>
7366     </wsdl:operation>
7367
7368     <wsdl:operation name="setTaskStartDurationExpression">
7369     <wsdl:input message="setTaskStartDurationExpression"/>
7370     <wsdl:output message="setTaskStartDurationExpressionResponse"/>
7371     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7372     <wsdl:fault name="illegalArgumentFault"
7373 message="illegalArgumentFault"/>
7374     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7375     <wsdl:fault name="illegalOperationFault"
7376 message="illegalOperationFault"/>
7377     </wsdl:operation>
7378
7379     <wsdl:operation name="skip">
7380     <wsdl:input message="skip"/>
7381     <wsdl:output message="skipResponse"/>
7382     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7383     <wsdl:fault name="illegalArgumentFault"
7384 message="illegalArgumentFault"/>
7385     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7386     <wsdl:fault name="illegalOperationFault"
7387 message="illegalOperationFault"/>
7388     </wsdl:operation>
7389
7390     <wsdl:operation name="batchSkip">
7391     <wsdl:input message="batchSkip"/>
7392     <wsdl:output message="batchSkipResponse"/>
7393     </wsdl:operation>
7394

```

```

7395     <wsdl:operation name="start">
7396         <wsdl:input message="start"/>
7397         <wsdl:output message="startResponse"/>
7398         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7399         <wsdl:fault name="illegalArgumentFault"
7400 message="illegalArgumentFault"/>
7401         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7402         <wsdl:fault name="illegalOperationFault"
7403 message="illegalOperationFault"/>
7404     </wsdl:operation>
7405
7406     <wsdl:operation name="batchStart">
7407         <wsdl:input message="batchStart"/>
7408         <wsdl:output message="batchStartResponse"/>
7409     </wsdl:operation>
7410
7411     <wsdl:operation name="stop">
7412         <wsdl:input message="stop"/>
7413         <wsdl:output message="stopResponse"/>
7414         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7415         <wsdl:fault name="illegalArgumentFault"
7416 message="illegalArgumentFault"/>
7417         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7418         <wsdl:fault name="illegalOperationFault"
7419 message="illegalOperationFault"/>
7420     </wsdl:operation>
7421
7422     <wsdl:operation name="batchStop">
7423         <wsdl:input message="batchStop"/>
7424         <wsdl:output message="batchStopResponse"/>
7425     </wsdl:operation>
7426
7427     <wsdl:operation name="suspend">
7428         <wsdl:input message="suspend"/>
7429         <wsdl:output message="suspendResponse"/>
7430         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7431         <wsdl:fault name="illegalArgumentFault"
7432 message="illegalArgumentFault"/>
7433         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7434         <wsdl:fault name="illegalOperationFault"
7435 message="illegalOperationFault"/>
7436     </wsdl:operation>
7437
7438     <wsdl:operation name="batchSuspend">
7439         <wsdl:input message="batchSuspend"/>
7440         <wsdl:output message="batchSuspendResponse"/>
7441     </wsdl:operation>
7442
7443     <wsdl:operation name="suspendUntil">
7444         <wsdl:input message="suspendUntil"/>
7445         <wsdl:output message="suspendUntilResponse"/>
7446         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7447         <wsdl:fault name="illegalArgumentFault"
7448 message="illegalArgumentFault"/>
7449         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7450         <wsdl:fault name="illegalOperationFault"
7451 message="illegalOperationFault"/>
7452     </wsdl:operation>

```

```

7453
7454     <wsdl:operation name="batchSuspendUntil">
7455         <wsdl:input message="batchSuspendUntil"/>
7456         <wsdl:output message="batchSuspendUntilResponse"/>
7457     </wsdl:operation>
7458
7459     <wsdl:operation name="updateComment">
7460         <wsdl:input message="updateComment"/>
7461         <wsdl:output message="updateCommentResponse"/>
7462         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7463         <wsdl:fault name="illegalArgumentFault"
7464 message="illegalArgumentFault"/>
7465         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7466         <wsdl:fault name="illegalOperationFault"
7467 message="illegalOperationFault"/>
7468     </wsdl:operation>
7469
7470     <wsdl:operation name="getMyTaskAbstracts">
7471         <wsdl:input message="getMyTaskAbstracts"/>
7472         <wsdl:output message="getMyTaskAbstractsResponse"/>
7473         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7474         <wsdl:fault name="illegalArgumentFault"
7475 message="illegalArgumentFault"/>
7476         <wsdl:fault name="illegalOperationFault"
7477 message="illegalOperationFault"/>
7478     </wsdl:operation>
7479
7480     <wsdl:operation name="getMyTaskDetails">
7481         <wsdl:input message="getMyTaskDetails"/>
7482         <wsdl:output message="getMyTaskDetailsResponse"/>
7483         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7484         <wsdl:fault name="illegalArgumentFault"
7485 message="illegalArgumentFault"/>
7486         <wsdl:fault name="illegalOperationFault"
7487 message="illegalOperationFault"/>
7488     </wsdl:operation>
7489
7490     <wsdl:operation name="query">
7491         <wsdl:input message="query"/>
7492         <wsdl:output message="queryResponse"/>
7493         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7494         <wsdl:fault name="illegalArgumentFault"
7495 message="illegalArgumentFault"/>
7496     </wsdl:operation>
7497
7498     <wsdl:operation name="activate">
7499         <wsdl:input message="activate"/>
7500         <wsdl:output message="activateResponse"/>
7501         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7502         <wsdl:fault name="illegalArgumentFault"
7503 message="illegalArgumentFault"/>
7504         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7505         <wsdl:fault name="illegalOperationFault"
7506 message="illegalOperationFault"/>
7507     </wsdl:operation>
7508
7509     <wsdl:operation name="batchActivate">
7510         <wsdl:input message="batchActivate"/>

```

```

7511     <wsdl:output message="batchActivateResponse"/>
7512 </wsdl:operation>
7513
7514     <wsdl:operation name="nominate">
7515         <wsdl:input message="nominate"/>
7516         <wsdl:output message="nominateResponse"/>
7517         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7518         <wsdl:fault name="illegalArgumentFault"
7519 message="illegalArgumentFault"/>
7520         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7521         <wsdl:fault name="illegalOperationFault"
7522 message="illegalOperationFault"/>
7523     </wsdl:operation>
7524
7525     <wsdl:operation name="batchNominate">
7526         <wsdl:input message="batchNominate"/>
7527         <wsdl:output message="batchNominateResponse"/>
7528     </wsdl:operation>
7529
7530     <wsdl:operation name="setGenericHumanRole">
7531         <wsdl:input message="setGenericHumanRole"/>
7532         <wsdl:output message="setGenericHumanRoleResponse"/>
7533         <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7534         <wsdl:fault name="illegalArgumentFault"
7535 message="illegalArgumentFault"/>
7536         <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7537         <wsdl:fault name="illegalOperationFault"
7538 message="illegalOperationFault"/>
7539     </wsdl:operation>
7540
7541     <wsdl:operation name="batchSetGenericHumanRole">
7542         <wsdl:input message="batchSetGenericHumanRole"/>
7543         <wsdl:output message="batchSetGenericHumanRoleResponse"/>
7544     </wsdl:operation>
7545
7546 </wsdl:portType>
7547 </wsdl:definitions>

```

7548

## E. WS-HumanTask Parent API Port Type

```

7549 <?xml version="1.0" encoding="UTF-8"?>
7550 <!--
7551 Copyright (c) OASIS Open 2009. All Rights Reserved.
7552 -->
7553 <wsdl:definitions
7554   targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
7555   humantask/leantask/api/200803"
7556   xmlns="http://docs.oasis-open.org/ns/bpel4people/ws-
7557   humantask/leantask/api/200803"
7558   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
7559   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
7560   xmlns:htd="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803"
7561   xmlns:htt="http://docs.oasis-open.org/ns/bpel4people/ws-
7562   humantask/types/200803">
7563
7564   <wsdl:documentation>
7565     Web Service Definition for WS-HumanTask 1.1 - Operations for Task Parent
7566     Applications
7567   </wsdl:documentation>
7568
7569   <wsdl:types>
7570     <xsd:schema
7571       targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
7572       humantask/leantask/api/200803"
7573       elementFormDefault="qualified"
7574       blockDefault="#all">
7575
7576       <xsd:import
7577         namespace="http://docs.oasis-open.org/ns/bpel4people/ws-
7578         humantask/200803"
7579         schemaLocation="ws-humantask.xsd"/>
7580       <xsd:import
7581         namespace="http://docs.oasis-open.org/ns/bpel4people/ws-
7582         humantask/types/200803"
7583         schemaLocation="ws-humantask-types.xsd"/>
7584
7585       <!-- Input and output elements -->
7586       <xsd:element name="registerLeanTaskDefinition">
7587         <xsd:complexType>
7588           <xsd:sequence>
7589             <xsd:element name="taskDefinition" type="htd:tLeanTask" />
7590           </xsd:sequence>
7591         </xsd:complexType>
7592       </xsd:element>
7593       <xsd:element name="registerLeanTaskDefinitionResponse">
7594         <xsd:complexType>
7595           <xsd:sequence>
7596             <xsd:element name="taskName" type="xsd:NCName" />
7597           </xsd:sequence>
7598         </xsd:complexType>
7599       </xsd:element>
7600
7601       <xsd:element name="unregisterLeanTaskDefinition">

```

```

7602     <xsd:complexType>
7603         <xsd:sequence>
7604             <xsd:element name="taskName" type="xsd:NCName" />
7605         </xsd:sequence>
7606     </xsd:complexType>
7607 </xsd:element>
7608 <xsd:element name="unregisterLeanTaskDefinitionResponse">
7609     <xsd:complexType>
7610         <xsd:sequence>
7611             <xsd:element name="taskName" type="xsd:NCName" />
7612         </xsd:sequence>
7613     </xsd:complexType>
7614 </xsd:element>
7615
7616 <xsd:element name="listLeanTaskDefinitions">
7617     <xsd:complexType>
7618         <xsd:sequence>
7619             <xsd:annotation>
7620                 <xsd:documentation>Empty message</xsd:documentation>
7621             </xsd:annotation>
7622         </xsd:sequence>
7623     </xsd:complexType>
7624 </xsd:element>
7625 <xsd:element name="listLeanTaskDefinitionsResponse">
7626     <xsd:complexType>
7627         <xsd:sequence>
7628             <xsd:element name="leanTaskDefinitions">
7629                 <xsd:complexType>
7630                     <xsd:sequence>
7631                         <xsd:element name="leanTaskDefinition" type="htd:tLeanTask"
7632 minOccurs="0" maxOccurs="unbounded" />
7633                     </xsd:sequence>
7634                 </xsd:complexType>
7635             </xsd:element>
7636         </xsd:sequence>
7637     </xsd:complexType>
7638 </xsd:element>
7639
7640 <xsd:element name="createLeanTask">
7641     <xsd:complexType>
7642         <xsd:sequence>
7643             <xsd:element name="inputMessage">
7644                 <xsd:complexType>
7645                     <xsd:sequence>
7646                         <xsd:any processContents="lax" namespace="##any" />
7647                     </xsd:sequence>
7648                 </xsd:complexType>
7649             </xsd:element>
7650             <xsd:element name="taskDefinition" type="htd:tLeanTask"
7651 minOccurs="0"/>
7652             <xsd:element name="taskName" type="xsd:NCName" minOccurs="0" />
7653         </xsd:sequence>
7654     </xsd:complexType>
7655 </xsd:element>
7656 <xsd:element name="createLeanTaskResponse">
7657     <xsd:complexType>
7658         <xsd:sequence>
7659             <xsd:element name="outputMessage">

```

```

7660         <xsd:complexType>
7661             <xsd:sequence>
7662                 <xsd:any processContents="lax" namespace="##any" />
7663             </xsd:sequence>
7664         </xsd:complexType>
7665     </xsd:element>
7666 </xsd:sequence>
7667 </xsd:complexType>
7668 </xsd:element>
7669
7670 <xsd:element name="createLeanTaskAsync">
7671     <xsd:complexType>
7672         <xsd:sequence>
7673             <xsd:element name="inputMessage">
7674                 <xsd:complexType>
7675                     <xsd:sequence>
7676                         <xsd:any processContents="lax" namespace="##any" />
7677                     </xsd:sequence>
7678                 </xsd:complexType>
7679             </xsd:element>
7680             <xsd:element name="taskDefinition" type="htd:tLeanTask"
7681 minOccurs="0"/>
7682             <xsd:element name="taskName" type="xsd:NCName" minOccurs="0" />
7683         </xsd:sequence>
7684     </xsd:complexType>
7685 </xsd:element>
7686 <xsd:element name="createLeanTaskAsyncResponse">
7687     <xsd:complexType>
7688         <xsd:sequence>
7689             <xsd:annotation>
7690                 <xsd:documentation>Empty message</xsd:documentation>
7691             </xsd:annotation>
7692         </xsd:sequence>
7693     </xsd:complexType>
7694 </xsd:element>
7695
7696 <xsd:element name="createLeanTaskAsyncCallback">
7697     <xsd:complexType>
7698         <xsd:sequence>
7699             <xsd:element name="outputMessage">
7700                 <xsd:complexType>
7701                     <xsd:sequence>
7702                         <xsd:any processContents="lax" namespace="##any" />
7703                     </xsd:sequence>
7704                 </xsd:complexType>
7705             </xsd:element>
7706         </xsd:sequence>
7707     </xsd:complexType>
7708 </xsd:element>
7709
7710 <!-- Fault elements -->
7711 <xsd:element name="illegalState">
7712     <xsd:complexType>
7713         <xsd:sequence>
7714             <xsd:element name="status" type="htt:tStatus"/>
7715             <xsd:element name="message" type="xsd:string"/>
7716         </xsd:sequence>
7717     </xsd:complexType>

```

```

7718     </xsd:element>
7719
7720     <xsd:element name="illegalArgument" type="xsd:string"/>
7721
7722     <xsd:element name="illegalAccess" type="xsd:string"/>
7723
7724     </xsd:schema>
7725 </wsdl:types>
7726
7727 <!-- Declaration of messages -->
7728 <wsdl:message name="registerLeanTaskDefinition">
7729   <wsdl:part name="registerLeanTaskDefinition"
7730 element="registerLeanTaskDefinition"/>
7731 </wsdl:message>
7732 <wsdl:message name="registerLeanTaskDefinitionResponse">
7733   <wsdl:part name="registerLeanTaskDefinitionResponse"
7734 element="registerLeanTaskDefinitionResponse"/>
7735 </wsdl:message>
7736
7737 <wsdl:message name="unregisterLeanTaskDefinition">
7738   <wsdl:part name="unregisterLeanTaskDefinition"
7739 element="unregisterLeanTaskDefinition"/>
7740 </wsdl:message>
7741 <wsdl:message name="unregisterLeanTaskDefinitionResponse">
7742   <wsdl:part name="unregisterLeanTaskDefinitionResponse"
7743 element="unregisterLeanTaskDefinitionResponse"/>
7744 </wsdl:message>
7745
7746 <wsdl:message name="listLeanTaskDefinitions">
7747   <wsdl:part name="listLeanTaskDefinitions"
7748 element="listLeanTaskDefinitions"/>
7749 </wsdl:message>
7750 <wsdl:message name="listLeanTaskDefinitionsResponse">
7751   <wsdl:part name="listLeanTaskDefinitionsResponse"
7752 element="listLeanTaskDefinitionsResponse"/>
7753 </wsdl:message>
7754
7755 <wsdl:message name="createLeanTask">
7756   <wsdl:part name="createLeanTask" element="createLeanTask"/>
7757 </wsdl:message>
7758 <wsdl:message name="createLeanTaskResponse">
7759   <wsdl:part name="createLeanTaskResponse"
7760 element="createLeanTaskResponse"/>
7761 </wsdl:message>
7762
7763 <wsdl:message name="createLeanTaskAsync">
7764   <wsdl:part name="createLeanTaskAsync" element="createLeanTaskAsync"/>
7765 </wsdl:message>
7766 <wsdl:message name="createLeanTaskAsyncResponse">
7767   <wsdl:part name="createLeanTaskAsyncResponse"
7768 element="createLeanTaskAsyncResponse"/>
7769 </wsdl:message>
7770
7771 <wsdl:message name="createLeanTaskAsyncCallback">
7772   <wsdl:part name="createLeanTaskAsyncCallback"
7773 element="createLeanTaskAsyncCallback"/>
7774 </wsdl:message>
7775

```

```

7776 <!-- Declaration of fault messages -->
7777 <wsdl:message name="illegalStateFault">
7778   <wsdl:part name="illegalState" element="illegalState"/>
7779 </wsdl:message>
7780 <wsdl:message name="illegalArgumentFault">
7781   <wsdl:part name="illegalArgument" element="illegalArgument"/>
7782 </wsdl:message>
7783 <wsdl:message name="illegalAccessFault">
7784   <wsdl:part name="illegalAccess" element="illegalAccess"/>
7785 </wsdl:message>
7786
7787 <!-- Port type definitions -->
7788 <wsdl:portType name="leanTaskOperations">
7789
7790   <wsdl:operation name="registerLeanTaskDefinition">
7791     <wsdl:input message="registerLeanTaskDefinition"/>
7792     <wsdl:output message="registerLeanTaskDefinitionResponse"/>
7793     <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
7794     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7795   </wsdl:operation>
7796
7797   <wsdl:operation name="unregisterLeanTaskDefinition">
7798     <wsdl:input message="unregisterLeanTaskDefinition"/>
7799     <wsdl:output message="unregisterLeanTaskDefinitionResponse"/>
7800     <wsdl:fault name="illegalArgumentFault"
7801 message="illegalArgumentFault"/>
7802     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7803   </wsdl:operation>
7804
7805   <wsdl:operation name="listLeanTaskDefinitions">
7806     <wsdl:input message="listLeanTaskDefinitions"/>
7807     <wsdl:output message="listLeanTaskDefinitionsResponse"/>
7808     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7809   </wsdl:operation>
7810
7811   <wsdl:operation name="createLeanTask">
7812     <wsdl:input message="createLeanTask"/>
7813     <wsdl:output message="createLeanTaskResponse"/>
7814     <wsdl:fault name="illegalArgumentFault"
7815 message="illegalArgumentFault"/>
7816     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7817   </wsdl:operation>
7818
7819   <wsdl:operation name="createLeanTaskAsync">
7820     <wsdl:input message="createLeanTaskAsync"/>
7821     <wsdl:output message="createLeanTaskAsyncResponse"/>
7822     <wsdl:fault name="illegalArgumentFault"
7823 message="illegalArgumentFault"/>
7824     <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
7825   </wsdl:operation>
7826
7827 </wsdl:portType>
7828
7829 <wsdl:portType name="leanTaskCallbackOperations">
7830
7831   <wsdl:operation name="createLeanTaskAsyncCallback">
7832     <wsdl:input message="createLeanTaskAsyncCallback"/>
7833   </wsdl:operation>

```

```
7834
7835     </wsdl:portType>
7836
7837 </wsdl:definitions>
```

## F. WS-HumanTask Protocol Handler Port Types

```

7839 <?xml version="1.0" encoding="UTF-8"?>
7840 <!--
7841 Copyright (c) OASIS Open 2009. All Rights Reserved.
7842 -->
7843 <wsdl:definitions
7844   targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
7845 humantask/protocol/200803"
7846   xmlns="http://docs.oasis-open.org/ns/bpel4people/ws-
7847 humantask/protocol/200803"
7848   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
7849   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
7850   xmlns:htp="http://docs.oasis-open.org/ns/bpel4people/ws-
7851 humantask/protocol/200803">
7852
7853   <wsdl:documentation>
7854     Web Service Definition for WS-HumanTask 1.1 - Operations WS-HumanTask
7855     Protocol Participants
7856   </wsdl:documentation>
7857
7858   <wsdl:types>
7859     <xsd:schema
7860       targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
7861 humantask/protocol/200803"
7862       elementFormDefault="qualified"
7863       blockDefault="#all">
7864
7865       <xsd:complexType name="tProtocolMsgType">
7866         <xsd:sequence>
7867           <xsd:any namespace="##other" processContents="lax" minOccurs="0"
7868             maxOccurs="unbounded" />
7869         </xsd:sequence>
7870         <xsd:anyAttribute namespace="##any" processContents="lax" />
7871       </xsd:complexType>
7872
7873       <xsd:element name="skipped" type="htp:tProtocolMsgType" />
7874       <xsd:element name="fault" type="htp:tProtocolMsgType" />
7875       <xsd:element name="exit" type="htp:tProtocolMsgType" />
7876
7877       <xsd:element name="responseAction" type="xsd:anyURI" />
7878       <xsd:element name="responseOperation" type="xsd:NCName" />
7879     </xsd:schema>
7880   </wsdl:types>
7881
7882   <wsdl:message name="skipped">
7883     <wsdl:part name="parameters" element="skipped" />
7884   </wsdl:message>
7885   <wsdl:message name="fault">
7886     <wsdl:part name="parameters" element="fault" />
7887   </wsdl:message>
7888   <wsdl:message name="exit">
7889     <wsdl:part name="parameters" element="exit" />
7890   </wsdl:message>
7891

```

```
7892
7893 <wsdl:portType name="clientParticipantPortType">
7894   <wsdl:operation name="skippedOperation">
7895     <wsdl:input message="skipped" />
7896   </wsdl:operation>
7897   <wsdl:operation name="faultOperation">
7898     <wsdl:input message="fault" />
7899   </wsdl:operation>
7900 </wsdl:portType>
7901
7902 <wsdl:portType name="humanTaskParticipantPortType">
7903   <wsdl:operation name="exitOperation">
7904     <wsdl:input message="exit" />
7905   </wsdl:operation>
7906 </wsdl:portType>
7907
7908 </wsdl:definitions>
```

7909

## G. WS-HumanTask Context Schema

```
7910 <?xml version="1.0" encoding="UTF-8"?>
7911 <!--
7912 Copyright (c) OASIS Open 2009. All Rights Reserved.
7913 -->
7914 <xsd:schema
7915   targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
7916   humantask/context/200803"
7917   xmlns="http://docs.oasis-open.org/ns/bpel4people/ws-
7918   humantask/context/200803"
7919   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
7920   xmlns:htt="http://docs.oasis-open.org/ns/bpel4people/ws-
7921   humantask/types/200803"
7922   elementFormDefault="qualified"
7923   blockDefault="#all">
7924
7925   <xsd:annotation>
7926     <xsd:documentation>
7927       XML Schema for WS-HumanTask 1.1 - Human Task Context for Task
7928       Interactions
7929     </xsd:documentation>
7930   </xsd:annotation>
7931
7932   <!-- other namespaces -->
7933   <xsd:import
7934     namespace="http://www.w3.org/XML/1998/namespace"
7935     schemaLocation="http://www.w3.org/2001/xml.xsd"/>
7936   <xsd:import
7937     namespace="http://docs.oasis-open.org/ns/bpel4people/ws-
7938     humantask/types/200803"
7939     schemaLocation="ws-humantask-types.xsd"/>
7940
7941   <!-- human task context -->
7942   <xsd:element name="humanTaskRequestContext"
7943     type="tHumanTaskRequestContext"/>
7944   <xsd:complexType name="tHumanTaskRequestContext">
7945     <xsd:complexContent>
7946       <xsd:extension base="tHumanTaskContextBase">
7947         <xsd:sequence>
7948           <xsd:element name="peopleAssignments" type="tPeopleAssignments"
7949             minOccurs="0"/>
7950           <xsd:element name="isSkipable" type="xsd:boolean" minOccurs="0"/>
7951           <xsd:element name="expirationTime" type="xsd:dateTime"
7952             minOccurs="0"/>
7953           <xsd:element name="activationDeferralTime" type="xsd:dateTime"
7954             minOccurs="0"/>
7955           <xsd:any namespace="##other" processContents="lax" minOccurs="0"
7956             maxOccurs="unbounded"/>
7957         </xsd:sequence>
7958       </xsd:extension>
7959     </xsd:complexContent>
7960   </xsd:complexType>
7961   <xsd:element name="humanTaskResponseContext"
7962     type="tHumanTaskResponseContext"/>
```

```

7963 <xsd:complexType name="tHumanTaskResponseContext">
7964 <xsd:complexContent>
7965 <xsd:extension base="tHumanTaskContextBase">
7966 <xsd:sequence>
7967 <xsd:element name="actualOwner" type="htt:tUser"/>
7968 <xsd:element name="actualPeopleAssignments"
7969 type="tPeopleAssignments"/>
7970 <xsd:element name="outcome" type="xsd:string" minOccurs="0"/>
7971 <xsd:any namespace="##other" processContents="lax" minOccurs="0"
7972 maxOccurs="unbounded"/>
7973 </xsd:sequence>
7974 </xsd:extension>
7975 </xsd:complexContent>
7976 </xsd:complexType>
7977 <xsd:complexType name="tHumanTaskContextBase" abstract="true">
7978 <xsd:sequence>
7979 <xsd:element name="priority" type="htt:tPriority" minOccurs="0"/>
7980 <xsd:element name="attachments" type="tAttachments" minOccurs="0"/>
7981 </xsd:sequence>
7982 </xsd:complexType>
7983
7984 <!-- people assignments -->
7985 <xsd:complexType name="tPeopleAssignments">
7986 <xsd:sequence>
7987 <xsd:element ref="genericHumanRole" minOccurs="0"
7988 maxOccurs="unbounded"/>
7989 </xsd:sequence>
7990 </xsd:complexType>
7991 <xsd:element name="genericHumanRole" type="tGenericHumanRole"
7992 abstract="true" block="restriction extension"/>
7993 <xsd:element name="potentialOwners" type="tGenericHumanRole"
7994 substitutionGroup="genericHumanRole"/>
7995 <xsd:element name="excludedOwners" type="tGenericHumanRole"
7996 substitutionGroup="genericHumanRole"/>
7997 <xsd:element name="taskInitiator" type="tGenericHumanRole"
7998 substitutionGroup="genericHumanRole"/>
7999 <xsd:element name="taskStakeholders" type="tGenericHumanRole"
8000 substitutionGroup="genericHumanRole"/>
8001 <xsd:element name="businessAdministrators" type="tGenericHumanRole"
8002 substitutionGroup="genericHumanRole"/>
8003 <xsd:element name="recipients" type="tGenericHumanRole"
8004 substitutionGroup="genericHumanRole"/>
8005 <xsd:complexType name="tGenericHumanRole">
8006 <xsd:sequence>
8007 <xsd:element ref="htt:organizationalEntity"/>
8008 </xsd:sequence>
8009 </xsd:complexType>
8010
8011 <!-- attachments -->
8012 <xsd:complexType name="tAttachments">
8013 <xsd:sequence>
8014 <xsd:element name="returnAttachments" type="tReturnAttachments"
8015 minOccurs="0"/>
8016 <xsd:element ref="htt:attachment" minOccurs="0" maxOccurs="unbounded"/>
8017 </xsd:sequence>
8018 </xsd:complexType>
8019 <xsd:simpleType name="tReturnAttachments">
8020 <xsd:restriction base="xsd:string">

```

```
8021     <xsd:enumeration value="all"/>
8022     <xsd:enumeration value="newOnly"/>
8023     <xsd:enumeration value="none"/>
8024     </xsd:restriction>
8025 </xsd:simpleType>
8026
8027 </xsd:schema>
```

8028

## H. WS-HumanTask Policy Assertion Schema

```
8029 <?xml version="1.0" encoding="UTF-8"?>
8030 <!--
8031 Copyright (c) OASIS Open 2009. All Rights Reserved.
8032 -->
8033 <xsd:schema
8034   targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
8035   humantask/policy/200803"
8036   xmlns="http://docs.oasis-open.org/ns/bpel4people/ws-
8037   humantask/policy/200803"
8038   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
8039   xmlns:wsp="http://www.w3.org/ns/ws-policy"
8040   elementFormDefault="qualified"
8041   blockDefault="#all">
8042
8043   <xsd:annotation>
8044     <xsd:documentation>
8045       XML Schema for WS-HumanTask 1.1 - WS-HumanTask Policy Assertion
8046     </xsd:documentation>
8047   </xsd:annotation>
8048
8049   <!-- other namespaces -->
8050   <xsd:import
8051     namespace="http://www.w3.org/ns/ws-policy"
8052     schemaLocation="http://www.w3.org/2007/02/ws-policy.xsd" />
8053
8054   <!-- ws-humantask policy assertion -->
8055   <xsd:element name="HumanTaskAssertion" type="tHumanTaskAssertion"/>
8056   <xsd:complexType name="tHumanTaskAssertion" >
8057     <xsd:attribute ref="wsp:Optional" />
8058     <xsd:anyAttribute namespace="##any" processContents="lax" />
8059   </xsd:complexType>
8060
8061 </xsd:schema>
```

8062

## I. Sample

8063

This appendix contains the full sample used in this specification.

8064

8065

### WSDL Definition

8066

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

8067

```
<!--
```

8068

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

8069

```
-->
```

8070

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

8071

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

8072

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

8073

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

8074

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

8075

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

8076

```
<wsdl:documentation>
```

8078

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

8079

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

8080

```
<wsdl:types>
```

8081

```
<xsd:schema
```

8082

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

8083

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

8084

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

8085

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

8086

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

8087

```
<xsd:complexType>
```

8088

```
<xsd:sequence>
```

8089

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

8090

```
<xsd:complexType>
```

8091

```
<xsd:sequence>
```

8092

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

8093

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

8094

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

8095

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

8096

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

8097

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

8098

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

8099

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

8100

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

8101

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

8102

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

8103

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

8104

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

8105

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

8106

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

8107

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

8108

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

8109

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

8110

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

8111

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

8112

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

```

8115 </wsdl:message>
8116 <wsdl:message name="ClaimApprovalResponse">
8117   <wsdl:part name="ClaimApprovalResponse" type="xsd:boolean" />
8118 </wsdl:message>
8119 <wsdl:message name="notifyRequest">
8120   <wsdl:part name="firstname" type="xsd:string" />
8121   <wsdl:part name="lastname" type="xsd:string" />
8122 </wsdl:message>
8123
8124 <wsdl:portType name="ClaimsHandlingPT">
8125   <wsdl:operation name="approve">
8126     <wsdl:input message="tns:ClaimApprovalRequest" />
8127   </wsdl:operation>
8128   <wsdl:operation name="escalate">
8129     <wsdl:input message="tns:ClaimApprovalRequest" />
8130   </wsdl:operation>
8131 </wsdl:portType>
8132
8133 <wsdl:portType name="ClaimsHandlingCallbackPT">
8134   <wsdl:operation name="approvalResponse">
8135     <wsdl:input message="tns:ClaimApprovalResponse" />
8136   </wsdl:operation>
8137 </wsdl:portType>
8138
8139 <wsdl:portType name="ClaimApprovalReminderPT">
8140   <wsdl:operation name="notify">
8141     <wsdl:input message="tns:notifyRequest" />
8142   </wsdl:operation>
8143 </wsdl:portType>
8144
8145 </wsdl:definitions>

```

8146

## 8147 Human Interaction Definition

```

8148 <?xml version="1.0" encoding="UTF-8"?>
8149 <!--
8150 Copyright (c) OASIS Open 2009. All Rights Reserved.
8151 -->
8152 <htd:humanInteractions
8153   xmlns:htd="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803"
8154   xmlns:htt="http://docs.oasis-open.org/ns/bpel4people/ws-
8155   humantask/types/200803"
8156   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
8157   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
8158   xmlns:cl="http://www.example.com/claims/"
8159   xmlns:tns="http://www.example.com"
8160   targetNamespace="http://www.example.com"
8161   xsi:schemaLocation="http://docs.oasis-open.org/ns/bpel4people/ws-
8162   humantask/200803 ../../xml/ws-humantask.xsd">
8163
8164   <htd:documentation>
8165     Example for WS-HumanTask 1.1 - WS-HumanTask Task Definition
8166   </htd:documentation>
8167
8168   <htd:import importType="http://schemas.xmlsoap.org/wsdl/"
8169     location="ws-humantask-example-claim-approval.wsdl"
8170     namespace="http://www.example.com/claims/" />
8171

```

```

8172 <htd:logicalPeopleGroups>
8173
8174   <htd:logicalPeopleGroup name="regionalClerks">
8175     <htd:documentation xml:lang="en-US">
8176       The group of clerks responsible for the region specified.
8177     </htd:documentation>
8178     <htd:parameter name="region" type="xsd:string" />
8179   </htd:logicalPeopleGroup>
8180
8181   <htd:logicalPeopleGroup name="regionalManager">
8182     <htd:documentation xml:lang="en-US">
8183       The manager responsible for the region specified.
8184     </htd:documentation>
8185     <htd:parameter name="region" type="xsd:string" />
8186   </htd:logicalPeopleGroup>
8187
8188   <htd:logicalPeopleGroup name="clerksManager">
8189     <htd:documentation xml:lang="en-US">
8190       The manager of the clerk whose user ID is passed as parameter.
8191     </htd:documentation>
8192     <htd:parameter name="clerkUserID" type="xsd:string" />
8193   </htd:logicalPeopleGroup>
8194
8195   <htd:logicalPeopleGroup name="directorClaims">
8196     <htd:documentation xml:lang="en-US">
8197       The functional director responsible for claims processing.
8198     </htd:documentation>
8199   </htd:logicalPeopleGroup>
8200
8201 </htd:logicalPeopleGroups>
8202
8203 <htd:tasks>
8204
8205   <htd:task name="ApproveClaim">
8206     <htd:documentation xml:lang="en-US">
8207       This task is used to handle claims that require manual
8208       approval.
8209     </htd:documentation>
8210
8211     <htd:interface portType="cl:ClaimsHandlingPT"
8212       operation="approve"
8213       responsePortType="cl:ClaimsHandlingCallbackPT"
8214       responseOperation="approvalResponse" />
8215
8216     <htd:priority>
8217       htd:getInput("ClaimApprovalRequest")/prio
8218     </htd:priority>
8219
8220     <htd:peopleAssignments>
8221       <htd:potentialOwners>
8222         <htd:from logicalPeopleGroup="regionalClerks">
8223           <htd:argument name="region">
8224             htd:getInput("ClaimApprovalRequest")/region
8225           </htd:argument>
8226         </htd:from>
8227       </htd:potentialOwners>
8228
8229     <htd:businessAdministrators>

```

```

8230     <htd:from logicalPeopleGroup="regionalManager">
8231         <htd:argument name="region">
8232             htd:getInput("ClaimApprovalRequest")/region
8233         </htd:argument>
8234     </htd:from>
8235 </htd:businessAdministrators>
8236 </htd:peopleAssignments>
8237
8238 <htd:delegation potentialDelegates="nobody" />
8239
8240 <htd:presentationElements>
8241
8242     <htd:name xml:lang="en-US">Approve Claim</htd:name>
8243     <htd:name xml:lang="de-DE">
8244         Genehmigung der Schadensforderung
8245     </htd:name>
8246
8247     <htd:presentationParameters>
8248         <htd:presentationParameter name="firstname"
8249             type="xsd:string">
8250             htd:getInput("ClaimApprovalRequest")/cust/firstname
8251         </htd:presentationParameter>
8252         <htd:presentationParameter name="lastname"
8253             type="xsd:string">
8254             htd:getInput("ClaimApprovalRequest")/cust/lastname
8255         </htd:presentationParameter>
8256         <htd:presentationParameter name="euroAmount"
8257             type="xsd:double">
8258             htd:getInput("ClaimApprovalRequest")/amount
8259         </htd:presentationParameter>
8260     </htd:presentationParameters>
8261
8262     <htd:subject xml:lang="en-US">
8263         Approve the insurance claim for €$euroAmount$ on behalf of
8264         $firstname$ $lastname$
8265     </htd:subject>
8266     <htd:subject xml:lang="de-DE">
8267         Genehmigung der Schadensforderung über €$euroAmount$ für
8268         $firstname$ $lastname$
8269     </htd:subject>
8270
8271     <htd:description xml:lang="en-US" contentType="text/plain">
8272         Approve this claim following corporate guideline
8273         #4711.0815/7 ...
8274     </htd:description>
8275     <htd:description xml:lang="en-US" contentType="text/html">
8276         <![CDATA[
8277         <p>
8278             Approve this claim following corporate guideline
8279             <b>#4711.0815/7</b>
8280             ...
8281         </p>
8282         ]]>
8283     </htd:description>
8284     <htd:description xml:lang="de-DE" contentType="text/plain">
8285         Genehmigen Sie diese Schadensforderung entsprechend
8286         Richtlinie Nr. 4711.0815/7 ...
8287     </htd:description>

```

```

8288 <htd:description xml:lang="de-DE" contentType="text/html">
8289 <![CDATA[
8290 <p>
8291     Genehmigen Sie diese Schadensforderung entsprechend
8292     Richtlinie
8293     <b>Nr. 4711.0815/7</b>
8294     ...
8295 </p>
8296 ]]>
8297 </htd:description>
8298
8299 </htd:presentationElements>
8300
8301 <htd:deadlines>
8302
8303 <htd:startDeadline name="sendReminder">
8304 <htd:documentation xml:lang="en-US">
8305     If not started within 3 days, - escalation notifications
8306     are sent if the claimed amount is less than 10000 - to the
8307     task's potential owners to remind them or their todo - to
8308     the regional manager, if this approval is of high priority
8309     (0,1, or 2) - the task is reassigned to Alan if the
8310     claimed amount is greater than or equal 10000
8311 </htd:documentation>
8312 <htd:for>P3D</htd:for>
8313
8314 <htd:escalation name="reminder">
8315
8316 <htd:condition>
8317 <![CDATA[
8318     htd:getInput("ClaimApprovalRequest")/amount < 10000
8319 ]]>
8320 </htd:condition>
8321
8322 <htd:toParts>
8323 <htd:toPart name="firstname">
8324     htd:getInput("ClaimApprovalRequest","ApproveClaim")
8325     /firstname
8326 </htd:toPart>
8327 <htd:toPart name="lastname">
8328     htd:getInput("ClaimApprovalRequest","ApproveClaim")
8329     /lastname
8330 </htd:toPart>
8331 </htd:toParts>
8332
8333 <htd:localNotification
8334     reference="tns:ClaimApprovalReminder">
8335
8336 <htd:documentation xml:lang="en-US">
8337     Reuse the predefined notification
8338     "ClaimApprovalReminder". Overwrite the recipients with
8339     the task's potential owners.
8340 </htd:documentation>
8341
8342 <htd:peopleAssignments>
8343 <htd:recipients>
8344 <htd:from>

```

```

8346         htd:getPotentialOwners("ApproveClaim")
8347     </htd:from>
8348 </htd:recipients>
8349 </htd:peopleAssignments>
8350
8351 </htd:localNotification>
8352
8353 </htd:escalation>
8354
8355 <htd:escalation name="highPrio">
8356
8357     <htd:condition>
8358         <![CDATA[
8359             (htd:getInput("ClaimApprovalRequest")/amount < 10000
8360             && htd:getInput("ClaimApprovalRequest")/prio <= 2)
8361         ]]>
8362     </htd:condition>
8363
8364     <!-- task input implicitly passed to the notification -->
8365
8366     <htd:notification name="ClaimApprovalOverdue">
8367         <htd:documentation xml:lang="en-US">
8368             An inline defined notification using the approval data
8369             as its input.
8370         </htd:documentation>
8371
8372         <htd:interface portType="cl:ClaimsHandlingPT"
8373             operation="escalate" />
8374
8375         <htd:peopleAssignments>
8376             <htd:recipients>
8377                 <htd:from logicalPeopleGroup="regionalManager">
8378                     <htd:argument name="region">
8379                         htd:getInput("ClaimApprovalRequest")/region
8380                     </htd:argument>
8381                 </htd:from>
8382             </htd:recipients>
8383         </htd:peopleAssignments>
8384
8385         <htd:presentationElements>
8386             <htd:name xml:lang="en-US">
8387                 Claim approval overdue
8388             </htd:name>
8389             <htd:name xml:lang="de-DE">
8390                 Überfällige Schadensforderungsgenehmigung
8391             </htd:name>
8392         </htd:presentationElements>
8393
8394     </htd:notification>
8395
8396 </htd:escalation>
8397
8398 <htd:escalation name="highAmountReassign">
8399
8400     <htd:condition>
8401         <![CDATA[
8402             htd:getInput("ClaimApprovalRequest")/amount >= 10000
8403         ]]>

```

```

8404     </htd:condition>
8405
8406     <htd:reassignment>
8407         <htd:documentation>
8408             Reassign task to Alan if amount is greater than or
8409             equal 10000.
8410         </htd:documentation>
8411
8412         <htd:potentialOwners>
8413             <htd:from>
8414                 <htd:literal>
8415                     <htt:organizationalEntity>
8416                         <htt:user>Alan</htt:user>
8417                     </htt:organizationalEntity>
8418                 </htd:literal>
8419             </htd:from>
8420         </htd:potentialOwners>
8421
8422     </htd:reassignment>
8423
8424 </htd:escalation>
8425
8426 </htd:startDeadline>
8427
8428
8429 <htd:completionDeadline name="notifyManager">
8430     <htd:documentation xml:lang="en-US">
8431         When not completed within 3 hours after having been
8432         claimed, the manager of the clerk who claimed the activity
8433         is notified.
8434     </htd:documentation>
8435     <htd:for>PT3H</htd:for>
8436
8437     <htd:escalation name="delayedApproval">
8438
8439         <htd:notification name="ClaimApprovalOverdue">
8440             <htd:documentation xml:lang="en-US">
8441                 An inline defined notification using the approval data
8442                 as its input.
8443             </htd:documentation>
8444
8445             <htd:interface portType="cl:ClaimsHandlingPT"
8446                 operation="escalate" />
8447
8448             <htd:peopleAssignments>
8449                 <htd:recipients>
8450                     <htd:from logicalPeopleGroup="clerksManager">
8451                         <htd:argument name="clerkUserID">
8452                             htd:getActualOwner("ApproveClaim")
8453                         </htd:argument>
8454                     </htd:from>
8455                 </htd:recipients>
8456             </htd:peopleAssignments>
8457
8458             <htd:presentationElements>
8459                 <htd:name xml:lang="en-US">
8460                     Claim approval overdue
8461                 </htd:name>

```

```

8462         <htd:name xml:lang="de-DE">
8463             Überfällige Schadensforderungsgenehmigung
8464         </htd:name>
8465     </htd:presentationElements>
8466
8467 </htd:notification>
8468
8469 </htd:escalation>
8470 </htd:completionDeadline>
8471
8472 <htd:completionDeadline name="notifyDirector">
8473     <htd:documentation xml:lang="en-US">
8474         When not completed within 2 days after having been
8475         claimed, the functional director of claims processing is
8476         notified.
8477     </htd:documentation>
8478     <htd:for>P2D</htd:for>
8479
8480 <htd:escalation name="severelyDelayedApproval">
8481
8482     <htd:notification name="ClaimApprovalOverdue">
8483         <htd:documentation xml:lang="en-US">
8484             An inline defined notification using the approval data
8485             as its input.
8486         </htd:documentation>
8487
8488         <htd:interface portType="cl:ClaimsHandlingPT"
8489             operation="escalate" />
8490
8491         <htd:peopleAssignments>
8492             <htd:recipients>
8493                 <htd:from logicalPeopleGroup="directorClaims">
8494                     <htd:argument name="clerkUserID">
8495                         htd:getActualOwner("ApproveClaim")
8496                     </htd:argument>
8497                 </htd:from>
8498             </htd:recipients>
8499         </htd:peopleAssignments>
8500
8501     <htd:presentationElements>
8502         <htd:name xml:lang="en-US">
8503             Claim approval severely overdue
8504         </htd:name>
8505         <htd:name xml:lang="de-DE">
8506             Hochgradig überfällige Schadensforderungsgenehmigung
8507         </htd:name>
8508     </htd:presentationElements>
8509
8510 </htd:notification>
8511
8512 </htd:escalation>
8513 </htd:completionDeadline>
8514
8515 </htd:deadlines>
8516
8517 </htd:task>
8518
8519 </htd:tasks>

```

```

8520
8521 <htd:notifications>
8522
8523 <htd:notification name="ClaimApprovalReminder">
8524 <htd:documentation xml:lang="en-US">
8525     This notification is used to remind people of pending
8526     out-dated claim approvals. Recipients of this notification
8527     maybe overridden when it is referenced.
8528 </htd:documentation>
8529
8530 <htd:interface portType="cl:ClaimApprovalReminderPT"
8531     operation="notify" />
8532
8533 <htd:peopleAssignments>
8534 <htd:recipients>
8535 <htd:from>
8536 <htd:literal>
8537 <htt:organizationalEntity>
8538 <htt:user>Alan</htt:user>
8539 <htt:user>Dieter</htt:user>
8540 <htt:user>Frank</htt:user>
8541 <htt:user>Gerhard</htt:user>
8542 <htt:user>Ivana</htt:user>
8543 <htt:user>Karsten</htt:user>
8544 <htt:user>Matthias</htt:user>
8545 <htt:user>Patrick</htt:user>
8546 </htt:organizationalEntity>
8547 </htd:literal>
8548 </htd:from>
8549 </htd:recipients>
8550 </htd:peopleAssignments>
8551
8552 <htd:presentationElements>
8553
8554 <htd:name xml:lang="en-US">Approve Claim</htd:name>
8555 <htd:name xml:lang="de-DE">
8556     Genehmigung der Schadensforderung
8557 </htd:name>
8558
8559 <htd:presentationParameters>
8560 <htd:presentationParameter name="firstname"
8561     type="xsd:string">
8562     htd:getInput("firstname")
8563 </htd:presentationParameter>
8564 <htd:presentationParameter name="lastname"
8565     type="xsd:string">
8566     htd:getInput("lastname")
8567 </htd:presentationParameter>
8568 <htd:presentationParameter name="id" type="xsd:string">
8569     htd:getInput("taskId")
8570 </htd:presentationParameter>
8571 </htd:presentationParameters>
8572
8573 <htd:subject xml:lang="en-US">
8574     Claim approval for $firstname$, $lastname$ is overdue. See
8575     task $id$.
8576 </htd:subject>
8577

```

```
8578     </htd:presentationElements>
8579
8580     </htd:notification>
8581
8582     </htd:notifications>
8583
8584 </htd:humanInteractions>
```

---

## 8585 J. Acknowledgements

8586 The following individuals have participated in the creation of this specification and are gratefully  
8587 acknowledged:

8588

### 8589 **Members of the BPEL4People Technical Committee:**

8590 Phillip Allen, Microsoft Corporation

8591 Ashish Agrawal, Adobe Systems

8592 Mike Amend, BEA Systems, Inc.

8593 Stefan Baeuerle, SAP AG

8594 Charlton Barreto, Adobe Systems

8595 Justin Brunt, TIBCO Software Inc.

8596 Martin Chapman, Oracle Corporation

8597 Luc Clément, Active Endpoints, Inc.

8598 Manoj Das, Oracle Corporation

8599 Alireza Farhoush, TIBCO Software Inc.

8600 Mark Ford, Active Endpoints, Inc.

8601 Sabine Holz, SAP AG

8602 Dave Ings, IBM

8603 Gershon Janssen, Individual

8604 Diane Jordan, IBM

8605 Anish Karmarkar, Oracle Corporation

8606 Ulrich Keil, SAP AG

8607 Oliver Kieselbach, SAP AG

8608 Matthias Kloppmann, IBM

8609 Dieter König, IBM

8610 Marita Kruempelmann, SAP AG

8611 Frank Leymann, IBM

8612 Mark Little, Red Hat

8613 Alexander Malek, Microsoft Corporation

8614 Ashok Malhotra, Oracle Corporation

8615 Mike Marin, IBM

8616 Vinkesh Mehta, Deloitte Consulting LLP

8617 Jeff Mischkinsky, Oracle Corporation

8618 Ralf Mueller, Oracle Corporation

8619 Krasimir Nedkov, SAP AG

8620 Benjamin Notheis, SAP AG

8621 Michael Pellegrini, Active Endpoints, Inc.

8622 Hannah Petereit, SAP AG

8623 Gerhard Pfau, IBM

8624 Karsten Ploesser, SAP AG

8625 Ravi Rangaswamy, Oracle Corporation  
8626 Alan Rickayzen, SAP AG  
8627 Michael Rowley, BEA Systems, Inc.  
8628 Ron Ten-Hove, Sun Microsystems  
8629 Ivana Trickovic, SAP AG  
8630 Alessandro Triglia, OSS Nokalva  
8631 Claus von Riegen, SAP AG  
8632 Peter Walker, Sun Microsystems  
8633 Franz Weber, SAP AG  
8634 Prasad Yendluri, Software AG, Inc.

8635

8636 **WS-HumanTask 1.0 Specification Contributors:**

8637 Ashish Agrawal, Adobe  
8638 Mike Amend, BEA  
8639 Manoj Das, Oracle  
8640 Mark Ford, Active Endpoints  
8641 Chris Keller, Active Endpoints  
8642 Matthias Kloppmann, IBM  
8643 Dieter König, IBM  
8644 Frank Leymann, IBM  
8645 Ralf Müller, Oracle  
8646 Gerhard Pfau, IBM  
8647 Karsten Plösser, SAP  
8648 Ravi Rangaswamy, Oracle  
8649 Alan Rickayzen, SAP  
8650 Michael Rowley, BEA  
8651 Patrick Schmidt, SAP  
8652 Ivana Trickovic, SAP  
8653 Alex Yiu, Oracle  
8654 Matthias Zeller, Adobe

8655

8656 The following individuals have provided valuable input into the design of this specification: Dave Ings,  
8657 Diane Jordan, Mohan Kamath, Ulrich Keil, Matthias Kruse, Kurt Lind, Jeff Mischkinsky, Bhagat Nainani,  
8658 Michael Pellegrini, Lars Rueter, Frank Ryan, David Shaffer, Will Stallard, Cyrille Waguët, Franz Weber,  
8659 and Eric Wittmann.

---

## **~~K. Non-Normative Text~~**

8661

## L.K. Revision History

8662 [optional; should not be included in OASIS Standards]

8663

| 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<br>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,<br>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.<br>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<br>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<br>Resolution of Issue #6 applied to section 6.2<br>Resolution of Issue #15 applied to section 6.2<br>Formatting (Word Document Map) |
| CD-01-rev-4  | 2008-10-29 | Michael Rowley  | Resolution of Issue #2<br>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<br>Issue 50, sections 3, 4, 6, 7 (htd:→htt:)<br>Issue 55, section 2.5.2 (import type xsd)   |

| Revision    | Date       | Editor          | Changes Made  |
|-------------|------------|-----------------|---|
|             |            |                 | Issue 56, section 7.2 (tProtocolMsgType)<br>Issue 60, section 6.1.1 (API fault type)<br>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) –<br>removal of erroneous statement regarding<br>the source of the value for the<br>responseOperation |
| CD-02-rev-3 | 2009-02-22 | Michael Rowley  | Issue 44, section 6.1.1 plus ws-<br>humantask.xsd and ws-humantask-<br>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-<br>types.xsd updated)  |
| CD-03-rev4  | 2009-06-01 | Luc Clément     | Issue 65 – Incorporation of an HT<br>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-<br>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<br>Consistency issues in API data types<br>Text formatting in new sections                                  |
| CD-04-rev5  | 2009-06-29 | Ravi Rangaswamy | Issue 98, 99  |
| CD-05-rev0  | 2009-07-15 | Luc Clément     | Committee Draft 5   |
| CD-05-rev1  | 2009-07-15 | Luc Clément     | Issue 117   |

| Revision              | Date                       | Editor                      | Changes Made  |
|-----------------------|----------------------------|-----------------------------|---|
| CD-05-rev2            | 2009-07-18                 | Dieter König                | Issue 100, 112, 115<br>Issue 79 revisited: task/leanTask schema   |
| CD-05-rev3            | 2009-08-06                 | Dieter König                | Issue 88, 101, 102, 113, 116, 119, 120,<br>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<br>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<br>XML artifacts copied back to appendix  |
| CD-05-rev11           | 2009-11-01                 | Luc Clément                 | Issues 130, 131, 132<br>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,<br>141, 142, 143<br>Editorial:<br>-- Sorted several operation lists/tables<br>(API operations and XPath functions)<br>-- Copied modified XML artifacts back to<br>appendix |
| CD-07                 | 2010-03-03                 | Luc Clément                 | Creating of CD07, Copyright date updates<br>and cover page annotation as Public<br>Review 02  |
| <a href="#">CD-08</a> | <a href="#">2010-04-14</a> | <a href="#">Luc Clément</a> | <a href="#">CD08</a>  |
| <a href="#">CD-09</a> | <a href="#">2010-04-14</a> | <a href="#">Luc Clément</a> | <a href="#">CD09 / PRD 03</a>   |

8664