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

Committee Draft 06 / Public Review Draft 01

04 November 2009

Specification URIs:

This Version:

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

Previous Version:

N/A

Latest Version:

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

Technical Committee:

OASIS BPEL4People TC

Chair:

Dave Ings, IBM

Editors:

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

Related work:

This specification is related to:

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

Declared XML Namespaces:

htd – http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803

hta – http://docs.oasis-open.org/ns/bpel4people/ws-humantask/api/200803

htlt - http://docs.oasis-open.org/ns/bpel4people/ws-humantask/leantask/api/200803

htt – http://docs.oasis-open.org/ns/bpel4people/ws-humantask/types/200803

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

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

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

Abstract:

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

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

Status:

Notices

Table of Contents

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

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

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

1 Introduction

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

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

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

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

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

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

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

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

1.1 Terminology

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

1.2 Normative References

[RFC 1766]

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

[RFC 2046]

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

[RFC 2119]

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

[RFC 2396]

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

[RFC 3066]

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

[WSDL 1.1]

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

[WS-Addr-Core]

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

[WS-Addr-SOAP]

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

[WS-Addr-WSDL]

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

[WS-C]

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

[WS-Policy]

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

[WS-PolAtt]

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

[XML Infoset]

XML Information Set, W3C Recommendation, available via http://www.w3.org/TR/2001/REC-xml-infoset-20011024/

[XML Namespaces]

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

[XML Schema Part 1]

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

[XML Schema Part 2]

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

[XMLSpec]

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

[XPATH 1.0]

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

1.3 Non-Normative References

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

1.4 Conformance Targets

The following conformance targets are defined as part of this specification

- WS-HumanTask Definition
  A WS-HumanTask Definition is any artifact that complies with the human interaction schema and additional constraints defined in this document.
- WS-HumanTask Processor
  A WS-HumanTask Processor is any implementation that accepts a WS-HumanTask definition and executes the semantics as defined in this document.
- WS-HumanTask Parent
  A WS-HumanTask Parent is any implementation that supports the Interoperable Protocol for Advanced Interactions with Human Tasks as defined in this document.
- WS-HumanTask Client
  A WS-HumanTask Client is any implementation that uses the Programming Interfaces of the WS-HumanTask Processor.

1.5 Overall Architecture

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

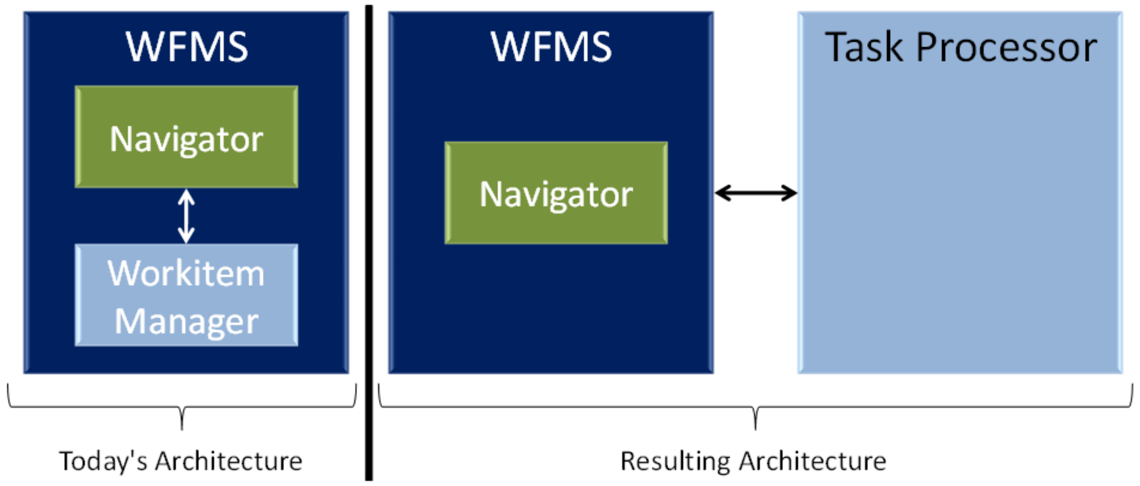


Figure 1- **Architectural Impact of WS-HumanTask on Workflow Management Systems**

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

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

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

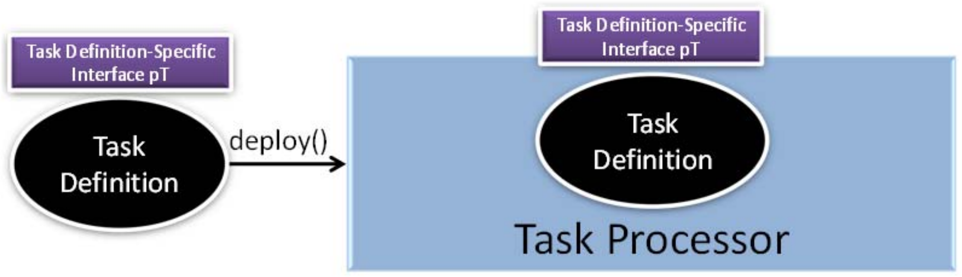

Task Definition-Specific Interface pT
Task Definition
deploy()
Task Definition-Specific Interface pT
Task Definition
Task Processor

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

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

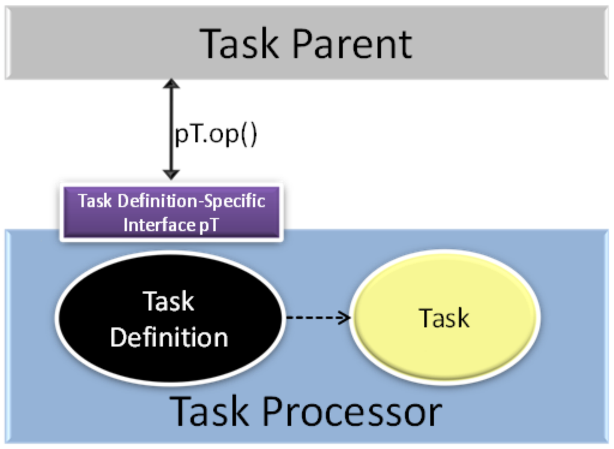


Figure 3 - **Instantiating Tasks**

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

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

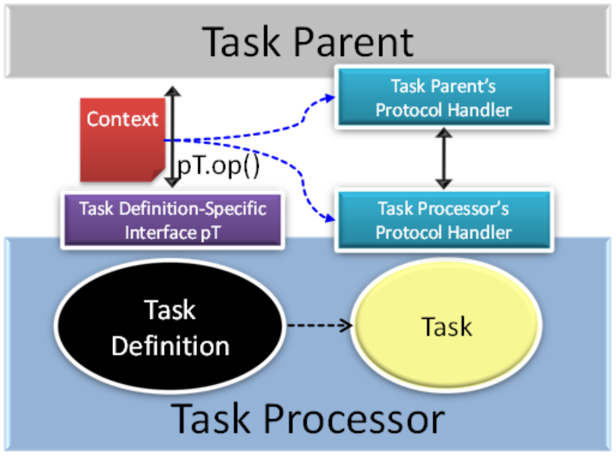

Task Parent
Task Parent's Protocol Handler
Context
pT.op()
Task Definition-Specific Interface pT
Task Processor's Protocol Handler
Task Definition
Task
Task Processor

Figure 4 - **Establishing a Protocol Channel**

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

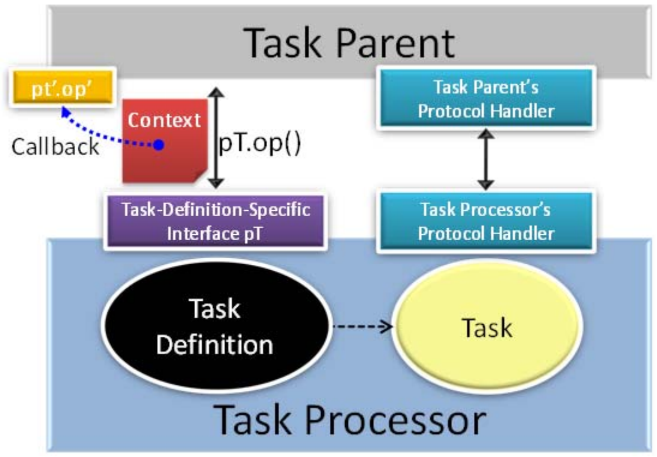

Task Parent
pt'.op'
Context
Callback
pT.op()
Task Parent's Protocol Handler
Task-Definition-Specific Interface pT
Task Processor's Protocol Handler
Task Definition
Task
Task Processor

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

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

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

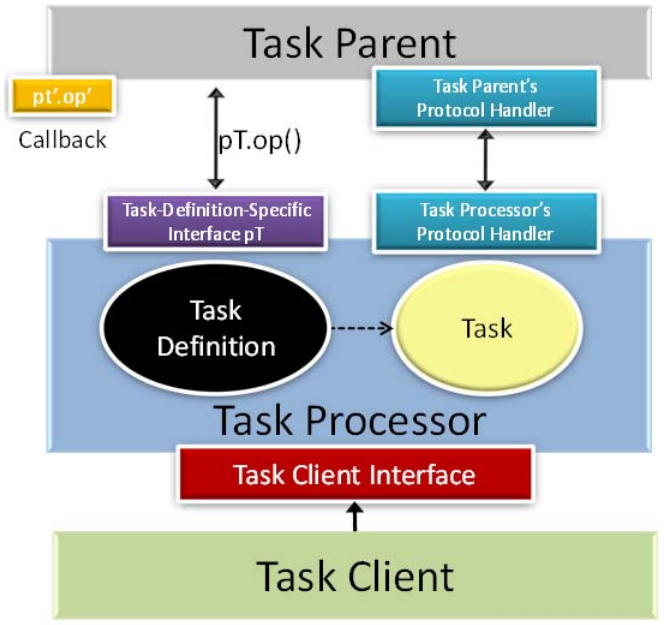

Task Parent
pt'.op'
Callback
pT.op()
Task Parent's Protocol Handler
Task-Definition-Specific Interface pT
Task Processor's Protocol Handler
Task Definition
Task
Task Processor
Task Client Interface
Task Client

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

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

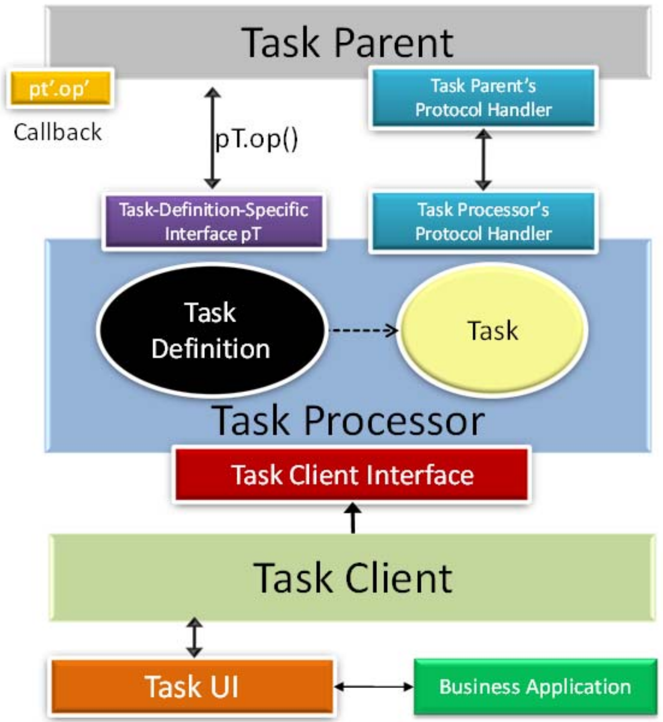


Figure 7 - **Overall Architecture of a Human Task Infrastructure**

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

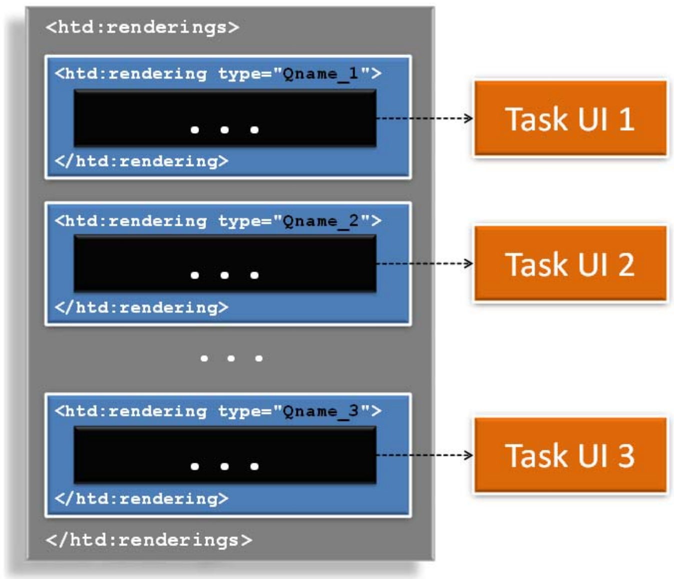
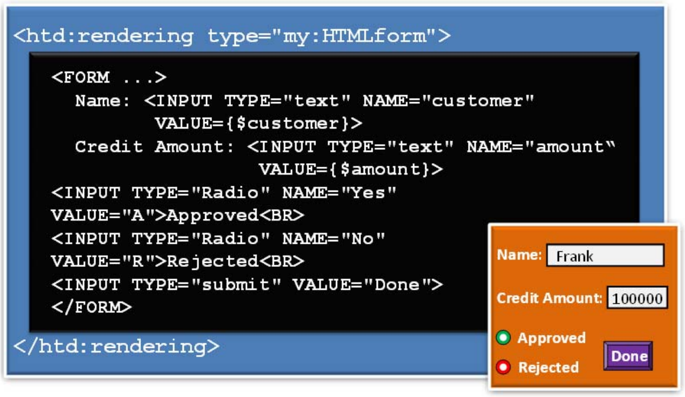


Figure 8 - **Potential Renderings of a Task**

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.



```
<htd:rendering type="my:HTMLform">
  <FORM ...>
    Name: <INPUT TYPE="text" NAME="customer"
           VALUE={$customer}>
    Credit Amount: <INPUT TYPE="text" NAME="amount"
                    VALUE={$amount}>
  <INPUT TYPE="Radio" NAME="Yes"
  VALUE="A">Approved<BR>
  <INPUT TYPE="Radio" NAME="No"
  VALUE="R">Rejected<BR>
  <INPUT TYPE="submit" VALUE="Done">
  </FORM>
</htd:rendering>
```

Name: Frank

Credit Amount: 100000

Figure 9 - **Sample Rendering of a Task**

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.

2 Language Design

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

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

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

2.1 Dependencies on Other Specifications

WS-HumanTask utilizes the following specifications:

- WSDL 1.1
- XML Schema 1.0
- XPath 1.0
- WS-Addressing 1.0
- WS-Coordination 1.1
- WS-Policy 1.5

2.1.1 Namespaces Referenced

WS-HumanTask references these namespaces:

- **wsa** – http://www.w3.org/2005/08/addressing
- **wsdl** – http://schemas.xmlsoap.org/wsdl/
- **wsp** – http://www.w3.org/ns/ws-policy
- **xsd** – http://www.w3.org/2001/XMLSchema

2.2 Language Extensibility

The WS-HumanTask extensibility mechanism allows:

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

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

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

2.3 Overall Language Structure

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

2.3.1 Syntax

```
<htd:humanInteractions
  xmlns:htd="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="anyURI"
  targetNamespace="anyURI"
  expressionLanguage="anyURI"?
  queryLanguage="anyURI"?>

  <htd:extensions>?
    <htd:extension namespace="anyURI" mustUnderstand="yes|no"/>+
  </htd:extensions>

  <htd:import namespace="anyURI"?
  location="anyURI"?
  importType="anyURI" />*

  <htd:logicalPeopleGroups>?
    <htd:logicalPeopleGroup name="NCName" reference="QName"?>+
      <htd:parameter name="NCName" type="QName" />*
    </htd:logicalPeopleGroup>
  </htd:logicalPeopleGroups>

  <htd:tasks>?
    <htd:task name="NCName">+
      ...
    </htd:task>
  </htd:tasks>

  <htd:notifications>?
    <htd:notification name="NCName">+
      ...
    </htd:notification>
  </htd:notifications>
</htd:humanInteractions>
```

2.3.2 Properties

The `<humanInteractions>` element has the following properties:

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

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

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

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

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

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

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

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

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

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

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

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

2.4 Default use of XPath 1.0 as an Expression Language

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

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

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

3 Concepts

3.1 Generic Human Roles

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

- Task initiator
- Task stakeholders
- Potential owners
- Actual owner
- Excluded owners
- Business administrators
- Notification recipients

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

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

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

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

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

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

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

3.2 Composite Tasks and Sub Tasks

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

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

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

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

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

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

3.2.1 Composite Tasks by Definition

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

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

3.2.2 Composite Tasks Created Adhoc at Runtime

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

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

3.3 Routing Patterns

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

3.4 Relationship of Composite Tasks and Routing Patterns

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

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

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

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

3.5 Assigning People

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

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

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

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.

Syntax:

```
<htd:peopleAssignments>

  <htd:genericHumanRole>+
    <htd:from>...</htd:from>
  </htd:genericHumanRole>

  <htd:potentialOwners>+
    fromPattern+
  </htd: potentialOwners>

</htd:peopleAssignments>
```

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

```
<htd:excludedOwners>
  <htd:from>...</htd:from>
</htd:excludedOwners>

<htd:taskInitiator>
  <htd:from>...</htd:from>
</htd:taskInitiator>
```

```

<htd:taskStakeholders>
  <htd:from>...</htd:from>
</htd:taskStakeholders>

<htd:businessAdministrators>
  <htd:from>...</htd:from>
</htd:businessAdministrators>

<htd:recipients>
  <htd:from>...</htd:from>
</htd:recipients>
```

For the potentialOwner generic human role the syntax is as following

```
<htd:potentialOwner>
  fromPattern+
</htd:potentialOwner>

where fromPattern is one of:

<htd:from> ... </htd:from>

<htd:sequence type="all|single"?>
  fromPattern*
</htd:sequence>

<htd:parallel type="all|single"?>
  fromPattern*
</htd:parallel>
```

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

3.5.1 Using Logical People Groups

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

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

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

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

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

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

Syntax:
```
<htd:from logicalPeopleGroup="NCName">
  <htd:argument name="NCName" expressionLanguage="anyURI"? >*
    expression
  </htd:argument>
</htd:from>
```

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

Example:
```
<htd:potentialOwners>
  <htd:from logicalPeopleGroup="regionalClerks">
    <htd:argument name="region">
      htd:getInput("part1")/region
    </htd:argument>
  </htd:from>
</htd:potentialOwners>
```

3.5.2 Using Literals

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

Syntax:

```
<htd:from>
  <htd:literal>
    ... literal value ...
  </htd:literal>
</htd:from>
```

Example specifying user identifiers:

```
<htd:potentialOwners>
  <htd:from>
    <htd:literal>
      <htt:organizationalEntity>
        <htt:user>Alan</htt:user>
        <htt:user>Dieter</htt:user>
        <htt:user>Frank</htt:user>
        <htt:user>Gerhard</htt:user>
        <htt:user>Ivana</htt:user>
        <htt:user>Karsten</htt:user>
        <htt:user>Matthias</htt:user>
        <htt:user>Patrick</htt:user>
      </htt:organizationalEntity>
    </htd:literal>
  </htd:from>
</htd:potentialOwners>
```

Example specifying group names:

```
<htd:potentialOwners>
  <htd:from>
    <htd:literal>
      <htt:organizationalEntity>
        <htt:group>bpel4people_authors</htt:group>
      </htt:organizationalEntity>
    </htd:literal>
  </htd:from>
</htd:potentialOwners>
```

3.5.3 Using Expressions

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

Syntax:

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

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

Example:

```
<htd:potentialOwners>
  <htd:from>htd:getInput("part1")/approvers</htd:from>
</htd:potentialOwners>

<htd:businessAdministrators>
  <htd:from>
    htd:except( htd:getInput("part1")/admins,
                htd:getInput("part1")/globaladmins[0] )
  </htd:from>
</htd:businessAdministrators>
```

3.5.4 Data Type for Organizational Entities

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

```
<xsd:element name="organizationalEntity" type="tOrganizationalEntity" />
<xsd:complexType name="tOrganizationalEntity">
  <xsd:choice maxOccurs="unbounded">
    <xsd:element name="user" type="tUser" />
    <xsd:element name="group" type="tGroup" />
  </xsd:choice>
</xsd:complexType>

<xsd:element name="user" type="tUser" />
<xsd:simpleType name="tUser">
  <xsd:restriction base="xsd:string" />
</xsd:simpleType>

<xsd:element name="group" type="tGroup" />
<xsd:simpleType name="tGroup">
  <xsd:restriction base="xsd:string" />
</xsd:simpleType>
```

3.5.5 Subtasks

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

- Task initiator
    a) Activation pattern "manual" → WS-HumanTask Processor MAY assign the actual owner of the composite task
    b) Activation pattern "automatic" → WS-HumanTask Processor MAY assign the initiator of the composite task
- Task stakeholders
    - A WS-HumanTask Processor MAY assign the actual owner of the composite task
- Potential owners
    - No default assignment (usually potential owners will explicitly be defined)
- Excluded owners

    - A WS-HumanTask Processor MUST assign the excluded owners of the composite task

      (This rule applies always, even though the excluded owners of a sub task may be enhanced by additional people)

- Business administrators
    - A WS-HumanTask Processor MAY assign the business administrators of the composite task

3.6 Task Rendering

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

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

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

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

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

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

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

3.7 Lean Tasks

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

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

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

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

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

3.8 Task Instance Data

Task instance data falls into three categories:

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

3.8.1 Presentation Data

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

3.8.2 Context Data

The task context includes the following:

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

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

3.8.3 Operational Data

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

3.8.3.1 Ad-hoc Attachments

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

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

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

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

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

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

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

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

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

Attachment Info Data Type

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

```
<xsd:element name="attachmentInfo" type="tAttachmentInfo" />
<xsd:complexType name="tAttachmentInfo">
  <xsd:sequence>
    <xsd:element name="identifier" type="xsd:anyURI" />
    <xsd:element name="name" type="xsd:string" />
    <xsd:element name="accessType" type="xsd:string" />
    <xsd:element name="contentType" type="xsd:string" />
    <xsd:element name="contentCategory" type="xsd:anyURI" />
    <xsd:element name="attachedTime" type="xsd:dateTime" />
    <xsd:element name="attachedBy" type="htt:tUser" />
    <xsd:any namespace="##other" processContents="lax"
             minOccurs="0" maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>
```

Attachment Data Type

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

```
<xsd:element name="attachment" type="tAttachment" />
<xsd:complexType name="tAttachment">
  <xsd:sequence>
    <xsd:element ref="attachmentInfo" />
    <xsd:element name="value" type="xsd:anyType" />
  </xsd:sequence>
</xsd:complexType>
```

3.8.3.2 Comments

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

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

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

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

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

Comment Data Type

The following data type is used to return comments.

```
<xsd:element name="comment" type="tComment" />
<xsd:complexType name="tComment">
  <xsd:sequence>
    <xsd:element name="id" type="xsd:string" />
    <xsd:element name="addedTime" type="xsd:dateTime" />
    <xsd:element name="addedBy" type="htt:tUser" />
    <xsd:element name="lastModifiedTime" type="xsd:dateTime" />
    <xsd:element name="lastModifiedBy" type="htd:tUser" />
    <xsd:element name="text" type="xsd:string" />
    <xsd:any namespace="##other" processContents="lax"
             minOccurs="0" maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>
```

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

3.8.4 Data Types for Task Instance Data

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

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

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

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

TaskAbstract Data Type

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

```
<xsd:complexType name="tTaskAbstract">
  <xsd:sequence>
    <xsd:element name="id"
                 type="xsd:string" />
    <xsd:element name="taskType"
                 type="xsd:string" />
    <xsd:element name="name"
                 type="xsd:QName" />
    <xsd:element name="status"
                 type="tStatus" />
    <xsd:element name="priority"
                 type="tPriority" minOccurs="0" />
    <xsd:element name="createdTime"
                 type="xsd:dateTime" />
    <xsd:element name="activationTime"
                 type="xsd:dateTime" minOccurs="0" />
    <xsd:element name="expirationTime"
                 type="xsd:dateTime" minOccurs="0" />
    <xsd:element name="isSkipable"
                 type="xsd:boolean" minOccurs="0" />
    <xsd:element name="hasPotentialOwners"
                 type="xsd:boolean" minOccurs="0" />
    <xsd:element name="startByTimeExists"
                 type="xsd:boolean" minOccurs="0" />
    <xsd:element name="completeByTimeExists"
                 type="xsd:boolean" minOccurs="0" />
    <xsd:element name="presentationName"
                 type="tPresentationName" minOccurs="0" />
    <xsd:element name="presentationSubject"
                 type="tPresentationSubject" minOccurs="0" />
    <xsd:element name="renderingMethodExists"
                 type="xsd:boolean" />
    <xsd:element name="hasOutput"
                 type="xsd:boolean" minOccurs="0" />
    <xsd:element name="hasFault"
                 type="xsd:boolean" minOccurs="0" />
    <xsd:element name="hasAttachments"
                 type="xsd:boolean" minOccurs="0" />
    <xsd:element name="hasComments"
                 type="xsd:boolean" minOccurs="0" />
    <xsd:element name="escalated"
                 type="xsd:boolean" minOccurs="0" />
    <xsd:element name="outcome"
                 type="xsd:string" minOccurs="0"/>
    <xsd:element name="parentTaskId"
                 type="xsd:string" minOccurs="0"/>
    <xsd:element name="hasSubTasks"
                 type="xsd:boolean" minOccurs="0"/>
    <xsd:any namespace="##other" processContents="lax"
             minOccurs="0" maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>
```

TaskDetails Data Type

```
<xsd:element name="taskDetails" type="tTaskDetails"/>
<xsd:complexType name="tTaskDetails">
  <xsd:sequence>
    <xsd:element name="id"
```

```
                    type="xsd:string"/>
    <xsd:element name="taskType"
                    type="xsd:string"/>
    <xsd:element name="name"
                    type="xsd:QName"/>
    <xsd:element name="status"
                    type="tStatus"/>
    <xsd:element name="priority"
                    type="htt:tPriority" minOccurs="0"/>
    <xsd:element name="taskInitiator"
                    type="htt:tUser" minOccurs="0"/>
    <xsd:element name="taskStakeholders"
                    type="htt:tOrganizationalEntity" minOccurs="0"/>
    <xsd:element name="potentialOwners"
                    type="htt:tOrganizationalEntity" minOccurs="0"/>
    <xsd:element name="businessAdministrators"
                    type="htt:tOrganizationalEntity" minOccurs="0"/>
    <xsd:element name="actualOwner"
                    type="htt:tUser" minOccurs="0"/>
    <xsd:element name="notificationRecipients"
                    type="htt:tOrganizationalEntity" minOccurs="0"/>
    <xsd:element name="createdTime"
                    type="xsd:dateTime"/>
    <xsd:element name="createdBy"
                    type="xsd:string" minOccurs="0"/>
    <xsd:element name="lastModifiedTime"
                    type="xsd:dateTime"/>
    <xsd:element name="lastModifiedBy"
                    type="xsd:string" minOccurs="0"/>
    <xsd:element name="activationTime"
                    type="xsd:dateTime" minOccurs="0"/>
    <xsd:element name="expirationTime"
                    type="xsd:dateTime" minOccurs="0"/>
    <xsd:element name="isSkipable"
                    type="xsd:boolean" minOccurs="0"/>
    <xsd:element name="hasPotentialOwners"
                    type="xsd:boolean" minOccurs="0"/>
    <xsd:element name="startByTimeExists"
                    type="xsd:boolean" minOccurs="0"/>
    <xsd:element name="completeByTimeExists"
                    type="xsd:boolean" minOccurs="0"/>
    <xsd:element name="presentationName"
                    type="tPresentationName" minOccurs="0"/>
    <xsd:element name="presentationSubject"
                    type="tPresentationSubject" minOccurs="0"/>
    <xsd:element name="renderingMethodExists"
                    type="xsd:boolean"/>
    <xsd:element name="hasOutput"
                    type="xsd:boolean" minOccurs="0"/>
    <xsd:element name="hasFault"
                    type="xsd:boolean" minOccurs="0"/>
    <xsd:element name="hasAttachments"
                    type="xsd:boolean" minOccurs="0"/>
    <xsd:element name="hasComments"
                    type="xsd:boolean" minOccurs="0"/>
    <xsd:element name="escalated"
                    type="xsd:boolean" minOccurs="0"/>
    <xsd:element name="searchBy"
```

```
                  type="xsd:string" minOccurs="0"/>
    <xsd:element name="outcome"
                  type="xsd:string" minOccurs="0"/>
    <xsd:element name="parentTaskId"
                  type="xsd:string" minOccurs="0"/>
    <xsd:element name="hasSubTasks"
                  type="xsd:boolean" minOccurs="0"/>
    <xsd:any namespace="##other" processContents="lax"
             minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

Common Data Types

```
<xsd:simpleType name="tPresentationName">
  <xsd:annotation>
    <xsd:documentation>length-restricted string</xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:string">
    <xsd:maxLength value="64" />
    <xsd:whiteSpace value="preserve" />
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="tPresentationSubject">
  <xsd:annotation>
    <xsd:documentation>length-restricted string</xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:string">
    <xsd:maxLength value="254" />
    <xsd:whiteSpace value="preserve" />
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="tStatus">
  <xsd:restriction base="xsd:string" />
</xsd:simpleType>

<xsd:simpleType name="tPredefinedStatus">
  <xsd:annotation>
    <xsd:documentation>for documentation only</xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="CREATED" />
    <xsd:enumeration value="READY" />
    <xsd:enumeration value="RESERVED" />
    <xsd:enumeration value="IN_PROGRESS" />
    <xsd:enumeration value="SUSPENDED" />
    <xsd:enumeration value="COMPLETED" />
    <xsd:enumeration value="FAILED" />
    <xsd:enumeration value="ERROR" />
    <xsd:enumeration value="EXITED" />
    <xsd:enumeration value="OBSOLETE" />
  </xsd:restriction>
</xsd:simpleType>
```

3.8.5 Sub Tasks

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

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

4 Human Tasks

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

4.1 Overall Syntax

Definition of human tasks:

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

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

  <htd:priority expressionLanguage="anyURI"? >?
    integer-expression
  </htd:priority>

  <htd:peopleAssignments>?
    ...
  </htd:peopleAssignments>

  <htd:completionBehavior>?
    ...
  </htd:completionBehavior>

  <htd:delegation
    potentialDelegatees="anybody|nobody|potentialOwners|other" />?
    <htd:from>?
      ...
    </htd:from>
  </htd:delegation>

  <htd:presentationElements>?
    ...
  </htd:presentationElements>

  <htd:possibleOutcomes>?
    ...
  </htd:possibleOutcomes>

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

  <htd:searchBy expressionLanguage="anyURI"? >?
    expression
  </htd:searchBy>

  <htd:renderings>?
    <htd:rendering type="QName">+
      ...
    </htd:rendering>
  </htd:renderings>

  <htd:deadlines>?
```

```

    <htd:startDeadline name="NCName">*
      ...
    </htd:startDeadline>

    <htd:completionDeadline name="NCName">*
      ...
    </htd:completionDeadline>

  </htd:deadlines>

  <htd:composition>?
    ...
  </htd:composition>

</htd:task>
```

4.2 Properties

The following attributes and elements are defined for tasks:

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

- `peopleAssignments`: This element is used to specify people assigned to different generic human roles, i.e. potential owners, and business administrator. The element is optional. See section 3.5 for more details on people assignments.
- `completionBehavior`: This element is used to specify completion conditions of the task. It is optional. See section 4.8 for more details on completion behavior.
- `delegation`: This element is used to specify constraints concerning delegation of the task. Attribute `potentialDelegatees` defines to whom the task can be delegated. One of the following values MUST be specified:
  - `anybody`: It is allowed to delegate the task to anybody
  - `potentialOwners`: It is allowed to delegate the task to potential owners previously selected
  - `other`: It is allowed to delegate the task to other people, e.g. authorized owners. The element `<from>` is used to determine the people to whom the task can be delegated.
  - `nobody`: It is not allowed to delegate the task.

  The delegation element is optional. If this element is not present the task is allowed to be delegated to anybody.
- `presentationElements`: This element is used to specify different information used to display the task in a task list, such as name, subject and description. See section 4.3 for more details on presentation elements. The element is optional.
- `outcome`: This optional element identifies the field (of an xsd simple type) in the output message which reflects the business result of the task. A conversion takes place to yield an outcome of type `xsd:string`. The optional attribute `queryLanguage` specifies the language used for selection. If not specified, the default language as inherited from the closest enclosing element that specifies the attribute is used.
- `searchBy`: This optional element is used to search for task instances based on a custom search criterion. The result of the expression evaluation is of type `xsd:string`. The `expressionLanguage` attribute specifies the language used in the expression. The attribute is optional. If not specified, the default language as inherited from the closest enclosing element that specifies the attribute is used.
- `rendering`: This element is used to specify the rendering method. It is optional. If not present, task rendering is implementation dependent. See section 4.4 for more details on rendering tasks.
- `deadlines`: This element specifies different deadlines. It is optional. See section 4.9 for more details on timeouts and escalations.
- `composition`: This element is used to specify subtasks of a composite task. It is optional. See section 4.6 for more details on composite tasks.

4.3 Presentation Elements

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

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

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

Human readable information can be specified in multiple languages.

Syntax:

```
<htd:presentationElements>

  <htd:name xml:lang="xsd:language"? >*
    Text
  </htd:name>

  <!-- For the subject and description only,
    replacement variables can be used. -->
  <htd:presentationParameters expressionLanguage="anyURI"? >?
    <htd:presentationParameter name="NCName" type="QName">+
      expression
    </htd:presentationParameter>
  </htd:presentationParameters>

  <htd:subject xml:lang="xsd:language"? >*
    Text
  </htd:subject>

  <htd:description xml:lang="xsd:language"?
                   contentType="mimeTypeString"? >*
    <xsd:any minOccurs="0" maxOccurs="unbounded" />
  </htd:description>

</htd:presentationElements>
```

Properties

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

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

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

Example:

```
<htd:presentationElements>

  <htd:name xml:lang="en-US">Approve Claim</htd:name>
  <htd:name xml:lang="de-DE">
    Genehmigung der Schadensforderung
  </htd:name>

  <htd:presentationParameters>
    <htd:presentationParameter name="firstname" type="xsd:string">
      htd:getInput("ClaimApprovalRequest")/cust/firstname
    </htd:presentationParameter>
    <htd:presentationParameter name="lastname" type="xsd:string">
      htd:getInput("ClaimApprovalRequest")/cust/lastname
    </htd:presentationParameter>
    <htd:presentationParameter name="euroAmount" type="xsd:double">
      htd:getInput("ClaimApprovalRequest")/amount
    </htd:presentationParameter>
  </htd:presentationParameters>

  <htd:subject xml:lang="en-US">
    Approve the insurance claim for €{$euroAmount} on behalf of
    {$firstname} {$lastname}
  </htd:subject>
  <htd:subject xml:lang="de-DE">
    Genehmigung der Schadensforderung über €{$euroAmount} für
    {$firstname} {$lastname}
  </htd:subject>

  <htd:description xml:lang="en-US" contentType="text/plain">
    Approve this claim following corporate guideline #4711.0815/7 ...
  </htd:description>
  <htd:description xml:lang="en-US" contentType="text/html">
    <p>
      Approve this claim following corporate guideline
      <b>#4711.0815/7</b>
      ...
    </p>
  </htd:description>
  <htd:description xml:lang="de-DE" contentType="text/plain">
    Genehmigen Sie diese Schadensforderung entsprechend Richtlinie Nr.
    4711.0815/7 ...
  </htd:description>
  <htd:description xml:lang="de-DE" contentType="text/html">
    <p>
      Genehmigen Sie diese Schadensforderung entsprechend Richtlinie
      <b>Nr. 4711.0815/7</b>
      ...
    </p>
  </htd:description>
```

```

</htd:presentationElements>
```

4.4 Task Possible Outcomes

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

```
<htd:possibleOutcomes>
  <htd:possibleOutcome name="NCName">+
    <htd:outcomeName xml:lang="xsd:language"?>+
      Language specific display
    </htd:outcomeName>
  </htd:possibleOutcome>
</htd:possibleOutcomes>
```

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

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

4.5 Elements for Rendering Tasks

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

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

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

Syntax:

```
<htd:renderings>
  <htd:rendering type="QName">+
    <xsd:any minOccurs="1" maxOccurs="1" />
  </htd:rendering>
</htd:renderings>
```

4.6 Elements for Composite Tasks

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

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

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

Syntax:

```
<htd:task>
  ...
  <htd:composition type="sequential|parallel"
                   instantiationPattern="manual|automatic">
    <htd:subtask name="NCName">+

      ( <htd:task>
          ...
        </htd:task>

      | <htd:localTask reference="QName">
          standard-overriding-elements
          ...
        </htd:localTask>
      )

       <htd:toParts>?
        <htd:toPart part="NCName" expressionLanguage="anyURI">+
          XPath expression
        </htd:toPart>
      </htd:toParts>

    </htd:subtask>

  </htd:composition>
  ...
</htd:task>
```

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

```
<htd:priority expressionLanguage="anyURI"? >
  integer-expression
</htd:priority>

<htd:peopleAssignments>?
  <htd:genericHumanRole>
    <htd:from>...</htd:from>
  </htd:genericHumanRole>
</htd:peopleAssignments>
```

4.7 Elements for People Assignment

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

Syntax:

```
<htd:peopleAssignments>

  <htd:potentialOwners>
    ...
  </htd:potentialOwners>

  <htd:excludedOwners>?
    ...
  </htd:excludedOwners>

  <htd:taskInitiator>?
    ...
  </htd:taskInitiator>

  <htd:taskStakeholders>?
    ...
  </htd:taskStakeholders>

  <htd:businessAdministrators>?
    ...
  </htd:businessAdministrators>
```

```

</htd:peopleAssignments>
```

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

Example:

```
<htd:peopleAssignments>
  <htd:potentialOwners>
    <htd:from logicalPeopleGroup="regionalClerks">
      <htd:argument name="region">
        htd:getInput("ClaimApprovalRequest")/region
      </htd:argument>
    </htd:from>
  </htd:potentialOwners>

  <htd:businessAdministrators>
    <htd:from logicalPeopleGroup="regionalManager">
      <htd:argument name="region">
        htd:getInput("ClaimApprovalRequest")/region
      </htd:argument>
    </htd:from>
  </htd:businessAdministrators>
</htd:peopleAssignments>
```

4.7.1 Routing Patterns

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

4.7.1.1 Parallel Pattern

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

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

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

Syntax:

```
<htd:potentialOwners>
  <htd:parallel type="all|single"?>
    <htd:completionBehavior>
    <htd:from>...</htd:from>*
    pattern*
  </htd:parallel>
</htd:potentialOwners>
```

Example:

```
<htd:peopleAssignments>
  <htd:potentialOwners>
    <htd:parallel type="all">
      <htd:from>
        htd:getInput("ClaimApprovalRequest")/claimAgent
      </htd:from>
    </htd:parallel>
  </htd:potentialOwners>
</htd:peopleAssignments>
```

4.7.1.2 Sequential Pattern

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

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

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

Syntax:

```
<htd:potentialOwners>
  <htd:sequence type="all|single"?>
    <htd:completionBehavior>?
    <htd:from>...</htd:from>*
    pattern*
  </htd:sequence>
</htd:potentialOwners>
```

Example:

```
<htd:peopleAssignments>
  <htd:potentialOwners>
    <htd:sequence type="all">
      <htd:from logicalPeopleGroup="regionalClerks">
        <htd:argument name="region">
          htd:getInput("ClaimApprovalRequest")/region
        </htd:argument>
      </htd:from>
      <htd:from logicalPeopleGroup="regionalManager">
        <htd:argument name="region">
          htd:getInput("ClaimApprovalRequest")/region
        </htd:argument>
      </htd:from>
    </htd:sequence>
  </htd:potentialOwners>
</htd:peopleAssignments/>
```

4.8 Completion Behavior

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

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

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

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

```
<htd:completionBehavior completionAction=”manual|automatic”?>?
  <htd:completion name="NCName">*
    <htd:condition ... >
      ...
    </htd:condition>
    <htd:result>?
      ...
    <htd:result>
  </htd:completion>
  <htd:defaultCompletion>?
    <htd:result>
      ...
    <htd:result>
  </htd:defaultCompletion>
</htd:completionBehavior>
```

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

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

4.8.1 Completion Conditions

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

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

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

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

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

The time functions that are available are defined as follows:

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

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

Completion conditions of a task MUST use only time functions.

4.8.1.1 Evaluating the Completion Condition

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

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

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

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

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

Example:

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

```
<htd:parallel>
  ...
  <htd:completionBehavior>
    <htd:completion>
      <htd:condition>
        htd:getInput("ClaimApprovalRequest")/amount < 1000
      </htd:condition>
      <htd:result> ... </htd:result>
    </htd:completion>
    <htd:completion>
      <htd:condition>
        ( htd:getCountOfSubtasksWithOutcome("Rejected") /
          htd:getCountOfSubtasks() > 0.5 )
        and htd:waitFor("P7D")
      </htd:condition>
      <htd:result> ... </htd:result>
    </htd:completion>
  </htd:completionBehavior>
  ...
</htd:parallel>
```

4.8.2 Result Construction from Parallel Subtasks

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

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

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

```
<htd:result>
  (
    <htd:aggregate ... />
  |
    <htd:copy> ... </htd:copy>
  )+
</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.

```
<htd:aggregate part="NCName"?
               location="query"?
               condition="bool-expr"?
               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:
    - o 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.
    - o 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) and optional arguments, in the following form:
  `FunctionName '(' ( Argument ( ',' Argument )* )? ')'`
  Important:
    - o 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.
    - o Within the `function` attribute, function arguments MUST be specified only for *additional* parameters defined for an aggregation function.

Example:

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

```
<element name="Award" type="tns:tAward" />
<complexType name="tAward">
  <sequence>
    <element name="AwardRecommended" type="xsd:string" />
    <element name="AwardDetails" type="tns:tAwardDetails" />
  </sequence>
</complexType>
<complexType name="tAwardDetails">
  <sequence>
    <element name="Amount" type="xsd:integer" />
    <element name="Appraisal" type="xsd:string" />
  </sequence>
```

```
</complexType>
```

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

```
<htd:parallel ...>
  ...
  <htd:completionBehavior>
    <htd:completion>
      <htd:condition> ... </htd:condition>
      <htd:result>
        <htd:aggregate location="/Award/AwardRecommended"
                       function="htd:mostFrequentOccurence()"/>
        <htd:aggregate location="/Award/AwardDetails/Amount"
                       condition="/Award/AwardRecommended='yes'"
                       function="htd:avg()"/>
        <htd:aggregate location="/Award/AwardDetails/Appraisal"
                       function="htd:concatWithDelimiter(',')"/>
      </htd:result>
    </htd:completion>
  </htd:completionBehavior>
</htd:parallel>
```

4.8.2.2 Explicit Result Assignment

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

```
<htd:copy>+
  <htd:from expressionLanguage="anyURI"?>
    expression
  </htd:from>
  <htd:to queryLanguage="anyURI"?>
    query
  </htd:to>
</htd:copy>
```

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

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

Example 1:

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

```
<element name="Order" type="tns:tOrder" />
<complexType name="tOrder">
  <sequence>
    <element name="Item" type="tns:tItem" maxOccurs="unbounded"/>
    <element name="TotalPrice" type="xsd:integer" />
  </sequence>
</complexType>
<complexType name="tItem">
  <sequence>
    ...
  </sequence>
</complexType>
```

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

```
<htd:parallel>
  ...
  <htd:completionBehavior>
    <htd:completion>
      <htd:condition> ... </htd:condition>
      <htd:result>
        <htd:copy>
          <htd:from>
            htd:getSubtaskOutputs("orderResponse", "/Order/Item")
         </htd:from>
          <htd:to>/Order/Item</htd:to>
        </htd:copy>
        <htd:copy>
          <htd:from>
            htd:sum(htd:getSubtaskOutputs("orderResponse",
                                          "/Order/TotalPrice"))
          </htd:from>
          <htd:to>/Order/TotalPrice</htd:to>
        </htd:copy>
      </htd:result>
    </htd:completion>
  </htd:completionBehavior>
</htd:parallel>
```

Example 2:

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

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

  <htd:composition type="parallel" ...>
    <htd:subtask name="bookHotel">
      <htd:task>
        ... produces hotelReservation ...
      </htd:task>
    </htd:subtask>
    <htd:subtask name="bookFlight">
      <htd:task>
        ... produces flightReservation ...
      </htd:task>
    </htd:subtask>
  </htd:composition>
  ...
  <htd:completionBehavior>
    <htd:defaultCompletion>
      <htd:result>
        <htd:copy>
          <htd:from>
            htd:getSubtaskOutput("bookHotel",
                                 "bookHotelResponse",
                                 "/hotelReservation")
          </htd:from>
          <htd:to>/itinerary/hotelReservation</htd:to>
        </htd:copy>
        <htd:copy>
          <htd:from>
            htd:getSubtaskOutput("bookFlight",
                                 "bookFlightResponse",
                                 "/flightReservation")
          </htd:from>
          <htd:to>/itinerary/flightReservation</htd:to>
        </htd:copy>
      </htd:result>
    </htd:defaultCompletion>
  </htd:completionBehavior>
</htd:task>
```

4.9 Elements for Handling Timeouts and Escalations

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:

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

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

Syntax:

```
<htd:deadlines>

  <htd:startDeadline name="NCName">*

    <htd:documentation xml:lang="xsd:language"? >*
      text
    </htd:documentation>

    ( <htd:for expressionLanguage="anyURI"? >
        duration-expression
      </htd:for>
    | <htd:until expressionLanguage="anyURI"? >
        deadline-expression
      </htd:until>
    )

    <htd:escalation name="NCName">*
      ...
    </htd:escalation>

  </htd:startDeadline>

  <htd:completionDeadline name="NCName">*
    ...
  </htd:completionDeadline>

</htd:deadlines>
```

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

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

Escalations are triggered if

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

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

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

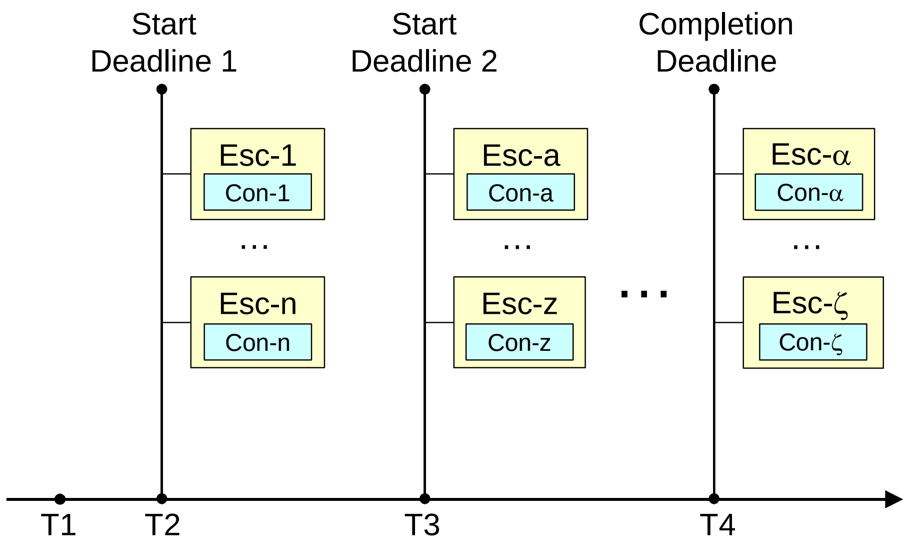
- A notification defined in the `<humanInteractions>` root element or imported from a different namespace can be referenced by specifying its QName in the `reference` attribute of a `<localNotification>` element. When referring to a notification, the priority and the people assignments of the original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

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

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

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

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



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

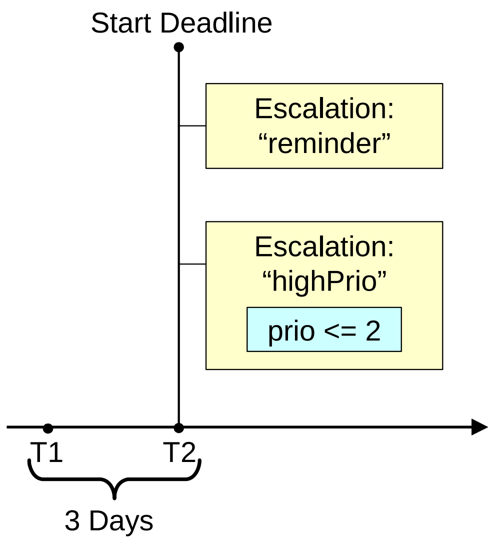
As an example, the figure depicts a task that has been created at time T1. Its two start deadlines would be missed at time T2 and T3, respectively. The associated escalations whose conditions evaluate to “true” are triggered. Both, the escalations Esc-1 to Esc-n as well as escalations Esc-a to Esc-z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

Syntax:

```
<htd:deadlines>

  <htd:startDeadline name="NCName">*
    ...
    <htd:escalation name="NCName">*

      <htd:condition expressionLanguage="anyURI"?>?
        boolean-expression
      </htd:condition>

      <htd:toParts>?
        <htd:toPart part="NCName"
                    expressionLanguage="anyURI"?>+
          expression
        </htd:toPart>
      </htd:toParts>

      <!--  notification specified by reference -->
      <htd:localNotification reference="QName">?
        <htd:priority expressionLanguage="anyURI"?>?
          integer-expression
        </htd:priority>
        <htd:peopleAssignments>?
          <htd:recipients>
            ...
          </htd:recipients>
        </htd:peopleAssignments>
      </htd:localNotification>

      <!--  notification specified inline -->
      <htd:notification name="NCName">?
        ...
      </htd:notification>

      <htd:reassignment>?

        <htd:potentialOwners>
          ...
        </htd:potentialOwners>

      </htd:reassignment>

    </htd:escalation>

  </htd:startDeadline>

  <htd:completionDeadline name="NCName">*
    ...
  </htd:completionDeadline>

</htd:deadlines>
```

Example:

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



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

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

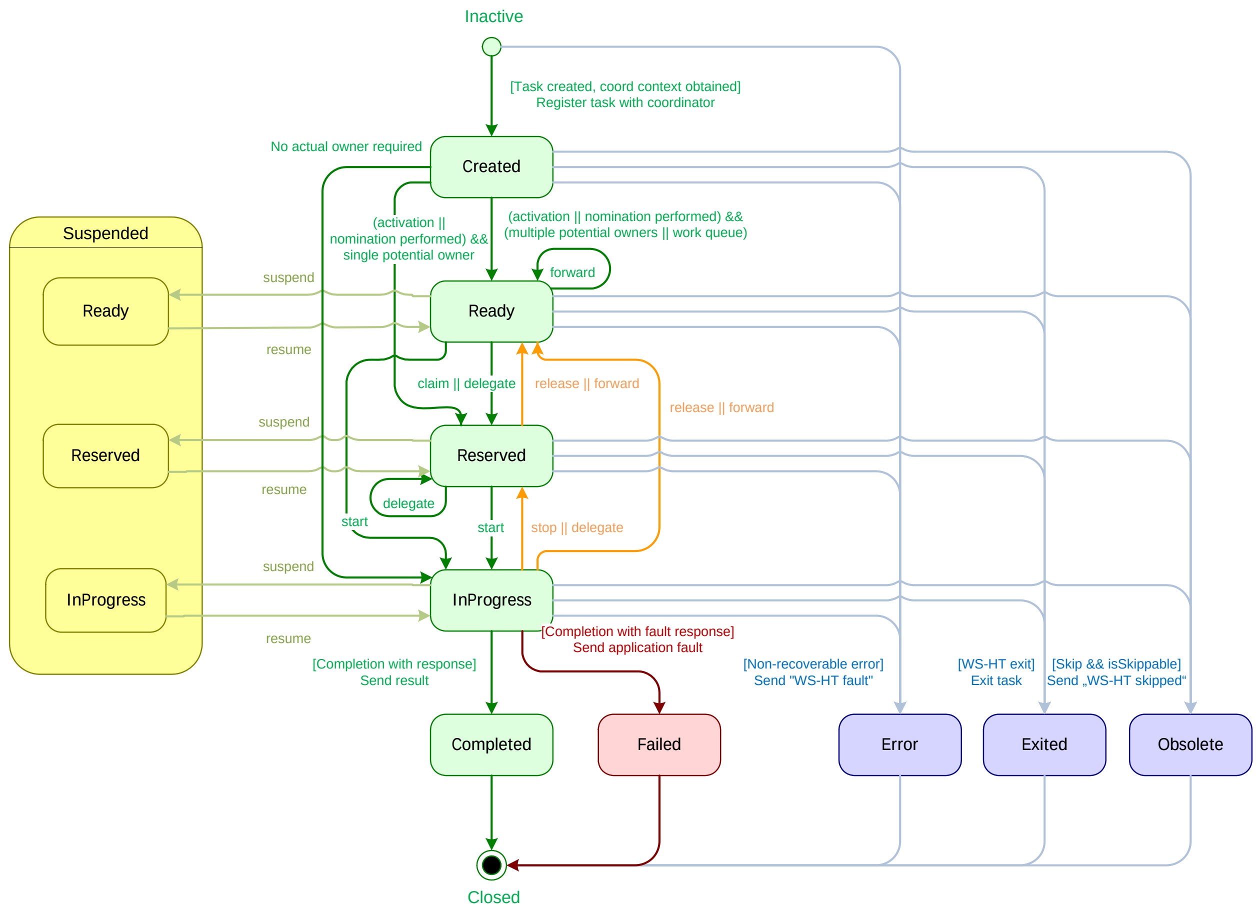
In case that task has been started before T2 was reached, then the start deadline is deactivated, no escalation occurs.

```
<htd:startDeadline name="sendNotifications">
  <htd:documentation xml:lang="en-US">
    If not started within 3 days, - escalation notifications are sent
    if the claimed amount is less than 10000 - to the task's potential
    owners to remind them or their todo - to the regional manager, if
    this approval is of high priority (0,1, or 2) - the task is
    reassigned to Alan if the claimed amount is greater than or equal
    10000
  </htd:documentation>
  <htd:for>P3D</htd:for>
  <htd:escalation name="reminder">

    <htd:condition>
      <![CDATA[
              htd:getInput("ClaimApprovalRequest")/amount < 10000
            ]]>
    </htd:condition>

    <htd:toParts>
      <htd:toPart name="firstname">
        htd:getInput("ClaimApprovalRequest","ApproveClaim")/firstname
      </htd:toPart>
      <htd:toPart name="lastname">
        htd:getInput("ClaimApprovalRequest","ApproveClaim")/lastname
      </htd:toPart>
    </htd:toParts>

    <htd:localNotification reference="tns:ClaimApprovalReminder">

      <htd:documentation xml:lang="en-US">
        Reuse the predefined notification "ClaimApprovalReminder".
        Overwrite the recipients with the task's potential owners.
      </htd:documentation>

      <htd:peopleAssignments>
        <htd:recipients>
          <htd:from>htd:getPotentialOwners("ApproveClaim")</htd:from>
        </htd:recipients>
      </htd:peopleAssignments>

    </htd:localNotification>

  </htd:escalation>

  <htd:escalation name="highPrio">

    <htd:condition>
      <![CDATA[
              (htd:getInput("ClaimApprovalRequest")/amount < 10000
            && htd:getInput("ClaimApprovalRequest")/prio <= 2)
            ]]>
    </htd:condition>

    <!-- task input implicitly passed to the notification -->

    <htd:notification name="ClaimApprovalOverdue">
      <htd:documentation xml:lang="en-US">
        An inline defined notification using the approval data as its
        input.
      </htd:documentation>

      <htd:interface portType="tns:ClaimsHandlingPT"
        operation="escalate" />

      <htd:peopleAssignments>
        <htd:recipients>
          <htd:from logicalPeopleGroup="regionalManager">
            <htd:argument name="region">
              htd:getInput("ClaimApprovalRequest")/region
            </htd:argument>
          </htd:from>
        </htd:recipients>
      </htd:peopleAssignments>

      <htd:presentationElements>
        <htd:name xml:lang="en-US">Claim approval overdue</htd:name>
        <htd:name xml:lang="de-DE">
          Überfällige Schadensforderungsgenehmigung
        </htd:name>
      </htd:presentationElements>

    </htd:notification>

  </htd:escalation>

  <htd:escalation name="highAmountReassign">

    <htd:condition>
      <![CDATA[
                htd:getInput("ClaimApprovalRequest")/amount >= 10000
              ]]>
    </htd:condition>

    <htd:reassignment>
      <htd:documentation>
        Reassign task to Alan if amount is greater than or equal
        10000.
      </htd:documentation>

      <htd:potentialOwners>
        <htd:from>
          <htd:literal>
            <htt:organizationalEntity>
              <htt:user>Alan</htt:user>
            </htt:organizationalEntity>
          </htd:literal>
        </htd:from>
      </htd:potentialOwners>

    </htd:reassignment>

  </htd:escalation>

</htd:startDeadline>
```

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

4.10 Human Task Behavior and State Transitions

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



4.10.1 Normal processing of a Human Task

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

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

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

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

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

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

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

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

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

1. Tasks with subtasks where an actual owner is required
2. Tasks with subtasks where no actual owner is required

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

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

4.10.2 Releasing a Human Task

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

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

4.10.3 Delegating or Forwarding a Human Task

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

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

4.10.4 Sub Task Event Propagation

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

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

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

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

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

4.11 History of a Human Task

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

Task Event

- event id
- event time
- task id
- user (principal) that caused the state change
- event type (e.g. claim task).
- event data (e.g. data used in setOutput) and fault name (event was setFault)
- startOwner - the actual owner before the event.
- endOwner - the actual owner after the event.
- task status at the end of the event

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

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

4.11.1 Task Event Types and Data

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

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

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

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

4.11.2 Retrieving the History

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

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

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

The XML Schema definition of the filter is the following:

```
<xsd:complexType name="tTaskHistoryFilter">
   <xsd:choice>
      <xsd:element name="eventId" type="xsd:integer" />
      <!--  Filter to allow narrow down query by status, principal,
            event Type. -->
      <xsd:sequence>
         <xsd:element name="status" type="tStatus" minOccurs="0"
           maxOccurs="unbounded" />
         <xsd:element name="eventType" type="tTaskEventType" minOccurs="0"
           maxOccurs="unbounded" />
         <xsd:element name="principal" type="xsd:string" minOccurs="0" />
         <xsd:element name="afterEventTime" type="xsd:dateTime"
           minOccurs="0" />
         <xsd:element name="beforeEventTime" type="xsd:dateTime"
           minOccurs="0" />
      </xsd:sequence>
   </xsd:choice>
</xsd:complexType>

<xsd:simpleType name="tTaskEventType">
   <xsd:restriction  base="xsd:string">
      <xsd:enumeration value="create" />
      <xsd:enumeration value="claim" />
      <xsd:enumeration value="start" />
      <xsd:enumeration value="stop" />
      <xsd:enumeration value="release" />
      <xsd:enumeration value="suspend" />
```

```
          <xsd:enumeration value="suspendUntil" />
          <xsd:enumeration value="resume" />
          <xsd:enumeration value="complete" />
          <xsd:enumeration value="remove" />
          <xsd:enumeration value="fail" />
          <xsd:enumeration value="setPriority" />
          <xsd:enumeration value="addAttachment" />
          <xsd:enumeration value="deleteAttachment" />
          <xsd:enumeration value="addComment" />
          <xsd:enumeration value="updateComment" />
          <xsd:enumeration value="deleteComment" />
          <xsd:enumeration value="skip" />
          <xsd:enumeration value="forward" />
          <xsd:enumeration value="delegate" />
          <xsd:enumeration value="setOutput" />
          <xsd:enumeration value="deleteOutput" />
          <xsd:enumeration value="setFault" />
          <xsd:enumeration value="deleteFault" />
          <xsd:enumeration value="activate" />
          <xsd:enumeration value="nominate" />
          <xsd:enumeration value="setGenericHumanRole" />
          <xsd:enumeration value="expire" />
          <xsd:enumeration value="escalated" />
       </xsd:restriction>
    </xsd:simpleType>
```

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

```
  <xsd:element name="taskEvent">
      <xsd:complexType>
         <xsd:annotation>
            <xsd:documentation>
               A detailed event that represents a change in the task's state.
            </xsd:documentation>
         </xsd:annotation>
         <xsd:sequence>
            <!--  event id - unique per task -->
            <xsd:element name="id" type="xsd:integer" />
            <!--  event  date time -->
            <xsd:element name="eventTime" type="xsd:dateTime" />
            <!--  task ID -->
            <xsd:element name="identifier" type="xsd:anyURI" />
            <xsd:element name="principal" type="xsd:string" minOccurs="0"
              nillable="true" />
            <!--  Event type. Note - using a restricted type limits
                  extensibility to add custom event types. -->
            <xsd:element name="eventType" type="tTaskEventType" />
            <!--  actual owner of the task before the event -->
            <xsd:element name="startOwner" type="xsd:string" minOccurs="0"
             nillable="true" />
            <!--  actual owner of the task after the event -->
            <xsd:element name="endOwner" type="xsd:string" minOccurs="0"
             nillable="true" />
            <!--  WSHT task status -->
            <xsd:element name="status" type="tStatus" />
            <!-- boolean to indicate this event has optional data -->
            <xsd:element name="hasData" type="xsd:boolean" minOccurs="0" />
            <xsd:element name="eventData" type="xsd:anyType" minOccurs="0"
             nillable="true" />
            <xsd:element name="faultName" type="xsd:string" minOccurs="0"
             nillable="true" />
            <!-- extensibility -->
            <xsd:any namespace="##other" processContents="lax" minOccurs="0"
             maxOccurs="unbounded" />
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
```

5 Lean Tasks

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

5.1 Overall Syntax

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

```
<htd:leanTask>
  ~~<htd:interface>….</htd:interface>~~
  <htd:messageSchema>...</htd:messageSchema>
  ... All elements from htd:task except <interface> and <composition> ...
  ~~<htd:composition>….</htd:composition>~~
</htd:leanTask>
```

5.2 Properties

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

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

5.3 Message Schema

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

```
<messageSchema>
  <messageField name="xsd:NCName" type="xsd:QName">*
    <messageDisplay xml:lang="xsd:language"?>+
      Language specific display
    </messageDisplay>
    <messageChoice name="xsd:NCName">*
      <messageDisplay xml:lang="xsd:language"?>+
        Language specific display
      </messageDisplay>
    </messageChoice>
  </messageField>
</messageSchema>
```

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

- String
- Integer

- Float
- Date Time
- Bool
- Enumeration (Choice)

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

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

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

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

Example:

```
<messageSchema>
  <messageField name="amount" type="xsd:float">
    <messageDisplay xml:lang="en-us">Amount</messageDisplay>
    <messageDisplay xml:lang="fr-fr”>Quantité</messageDisplay>
  </messageField>
  <messageField name="currencyUnit" type="xsd:string">
    <messageDisplay xml:lang="en-us">Currency<messageDisplay>
    <messageDisplay xml:lang="fr-fr">Devise</messageDisplay>
    <messageChoice name="USD">
      <messageDisplay xml:lang="en-us">US Dollars</messageDisplay>
      <messageDisplay xml:lang="fr-fr">US Dollars</messageDisplay>
    </messageChoice>
    <messageChoice name="EURO">
      <messageDisplay xml:lang="en-us">Euro Dollars</messageDisplay>
      <messageDisplay xml:lang="fr-fr">Euros</messageDisplay>
    </messageChoice>
  </messageField>
</messageSchema>
```

5.4 Example: ToDoTask

The following XML could be used for a simple ‘ToDoTask’:

```
<htd:task name="ToDoTask">
  <htd:messageSchema />
  <htd:possibleOutcomes>
    <htd:possibleOutcome name="Completed" />
      ... language specific translations ...
  </htd:possibleOutcomes>
  <htd:delegation potentialDelegates="anybody" />
  <htd:presentationElements>
    <htd:name>ToDo Task</htd:name>
      ... language specific translations ...
    <htd:subject>Please complete the described work</htd:subject>
      ... language specific translations ...
    <htd:description contentType="mimeTypeString" />
      ... language specific translations ...
  </htd:presentationElements>
</htd:task>
```

6 Notifications

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

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

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

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

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

6.1 Overall Syntax

Definition of notifications

```
<htd:notification name="NCName">

  <htd:interface portType="QName" operation="NCName"/>

  <htd:priority expressionLanguage="anyURI"?>?
    integer-expression
  </htd:priority>

  <htd:peopleAssignments>

    <htd:recipients>
      ...
    </htd:recipients>

    <htd:businessAdministrators>?
      ...
    </htd:businessAdministrators>

  </htd:peopleAssignments>

  <htd:presentationElements>
    ...
  </htd:presentationElements>

  <htd:renderings>?
```

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

</htd:notification>
```

6.2 Properties

The following attributes and elements are defined for notifications:

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

6.3 Notification Behavior and State Transitions

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

7 Programming Interfaces

7.1 Operations for Client Applications

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

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

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

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

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

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

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

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

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

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

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

7.1.1 Participant Operations

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

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

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

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

Operation Name	**Description**	**Parameters**	**Supports Batch Processing**	**Pre-State**	**Post-State**
claim	Claim responsibility for a task, i.e. set the task to status *Reserved*	In • task identifier Out • void	Yes	Ready	Reserved
start	Start the execution of the task, i.e. set the task to status *InProgress*.	In • task identifier Out • void	Yes	Ready Reserved	InProgress
stop	Cancel/stop the processing of the task. The task returns to the *Reserved* state.	In • task identifier Out • void	Yes	InProgress	Reserved
release	Release the task, i.e. set the task back to status *Ready*.	In • task identifier Out • void	Yes	InProgress Reserved	Ready
suspend	Suspend the task.	In • task identifier Out • void	Yes	Ready Reserved InProgress	Suspended/ Ready (from Ready) Suspended/ Reserved (from Reserved) Suspended/InProgress (from InProgress)
suspendUntil	Suspend the task for a given period of time or until a fixed point in time. The WS-HumanTask Client MUST specify either a period of time or a fixed point in time.	In • task identifier • time period • point of time Out • void	Yes	Ready Reserved InProgress	Suspended/ Ready (from Ready) Suspended/ Reserved (from Reserved) Suspended/InProgress (from InProgress)

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
resume	Resume a suspended task.	In • task identifier Out • void	Yes	Suspended/ Ready Suspended/ Reserved Suspended/InProgress	Ready (from Suspended/ Ready) Reserved (from Suspended/ Reserved) InProgress (from Suspended/InProgress)
complete	Execution of the task finished successfully. If no output data is set the operation MUST return `hta:illegalArgumentFault.`	In • task identifier • output data of task Out • void	Yes	InProgress	Completed
remove	Applies to notifications only. Used by notification recipients to remove the notification permanently from their task list client. It will not be returned on any subsequent retrieval operation invoked by the same user.	In • task identifier Out • void	Yes	Ready (Notification state)	Removed (Notification state)
fail	Execution of the task fails and a fault is returned. The fault `hta:illegalOperationFault` MUST be returned if the task interface defines no faults. If fault name or fault data is not set the operation MUST return `hta:illegalArgumentFault.`	In • task identifier • fault – contains the fault name and fault data Out • void	Yes	InProgress	Failed

Operation Name	**Description**	**Parameters**	**Supports Batch Processing**	**Pre-State**	**Post-State**
setPriority	Change the priority of the task. The WS-HumanTask Client MUST specify the integer value of the new priority.	In • task identifier • priority (`htt:tPriority`) Out • void	Yes	(any state)	(no state transition)
addAttachment	Add attachment to a task. Returns an identifier for the attachment.	In • task identifier • attachment name • access type • content type • attachment Out • attachment identifier	No	(any state)	(no state transition)
getAttachmentInfos	Get attachment information for all attachments associated with the task.	In • task identifier Out • list of attachment data (list of `htt:attachmentInfo`)	No	(any state)	(no state transition)
getAttachment	Get the task attachment with the given identifier.	In • task identifier • attachment identifier Out • `htt:attachment`	No	(any state)	(no state transition)
deleteAttachment	Delete the attachment with the specified identifier from the task. Attachments provided by the enclosing context MUST NOT be affected by this operation.	In • task identifier • attachment identifier Out • void	No	(any state)	(no state transition)

Operation Name	**Description**	**Parameters**	**Supports Batch Processing**	**Pre-State**	**Post-State**
addComment	Add a comment to a task. Returns an identifier that can be used to later update or delete the comment.	In • task identifier • plain text Out • comment identifier	No	(any state)	(no state transition)
updateComment	Updates the identified comment with the supplied new text.	In • task identifier • comment identifier • plain text Out • void	No	(any state)	(no state transition)
deleteComment	Deletes the identified comment.	In • task identifier • comment identifier Out • void	No	(any state)	(no state transition)
getComments	Get all comments of a task	In • task identifier Out • list of comments (list of `htt:comment`)	No	(any state)	(no state transition)
skip	Skip the task. If the task is not skipable then the fault `hta:illegalOperationFault` MUST be returned.	In • task identifier Out • void	Yes	Created Ready Reserved InProgress	Obsolete

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
forward	Forward the task to another organization entity. The WS-HumanTask Client MUST specify the receiving organizational entity. Potential owners MAY forward a task while the task is in the *Ready* state. For details on forwarding human tasks refer to section 4.10.3.	In • task identifier • organizational entity (`htt:tOrganizationalEntity`) Out • void	Yes	Ready Reserved InProgress	Ready
delegate	Assign the task to one user and set the task to state *Reserved*. If the recipient was not a potential owner then this person MUST be added to the set of potential owners. For details on delegating human tasks refer to section 4.10.3.	In • task identifier • organizational entity (`htt:tOrganizationalEntity`) Out • void	Yes	Ready Reserved InProgress	Reserved
getRendering	Applies to both tasks and notifications. Returns the rendering specified by the type parameter.	In • task identifier • rendering type Out • any type	No	(any state)	(no state transition)
getRenderingTypes	Applies to both tasks and notifications. Returns the rendering types available for the task or notification.	In • task identifier Out • list of QNames	No	(any state)	(no state transition)
getTaskDetails	Applies to both tasks and notifications. Returns a data object of type `htt:tTaskDetails`	In • task identifier Out • task (`htt:tTaskDetails`)	No	(any state)	(no state transition)

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
getTaskDescription	Applies to both tasks and notifications. Returns the presentation description in the specified mime type.	In • task identifier • content type – optional, default is text/plain Out • string	No	(any state)	(no state transition)
getTaskOperations	Applies to tasks. Returns list of operations that are available to the authorized user given the user's role and the state of the task.	In • task identifier Out • List of available operation.	No	(any state)	(no state transition)
setOutput	Set the data for the part of the task's output message.	In • task identifier • part name (optional for single part messages) • output data of task Out • void	No	InProgress	(no state transition)
deleteOutput	Deletes the output data of the task.	In • task identifier Out • void	No	InProgress	(no state transition)
setFault	Set the fault data of the task. The fault `hta:illegalOperationFault` MUST be returned if the task interface defines no faults.	In • task identifier • fault – contains the fault name and fault data Out • void	No	InProgress	(no state transition)
deleteFault	Deletes the fault name and fault data of the task.	In • task identifier Out • void	No	InProgress	(no state transition)

Operation Name	**Description**	**Parameters**	**Supports Batch Processing**	**Pre-State**	**Post-State**
getInput	Get the data for the part of the task's input message.	In • task identifier • part name (optional for single part messages) Out • any type	No	(any state)	(no state transition)
getOutput	Get the data for the part of the task's output message.	In • task identifier • part name (optional for single part messages) Out • any type	No	(any state)	(no state transition)
getFault	Get the fault data of the task.	In • task identifier Out • fault – contains the fault name and fault data	No	(any state)	(no state transition)
getOutcome	Get the outcome of the task	In • task identifier Out • string	No	(any state)	(no state transition)

Operation Name	**Description**	**Parameters**	**Supports Batch Processing**	**Pre-State**	**Post-State**
getTaskHistory	Get a list of events representing the history of the task. *Filter* allows narrowing the results by status, principal, event Type. *startIndex* and *maxTasks* are integers that allow paging of the results. *includeData* is a Boolean. Data is included with the returned events only if this is true.	In • task identifier • filter (`htt:tTaskHistoryFilter`) • startIndex • maxTasks • includeData Out • list of htt:taskEvent	No	(any state)	(no state transition)

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
getTaskInstance Data	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. *Properties* 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. If it is not specified, then all properties are returned. 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. Some properties of a task may have multiple values (i.e., taskDescription, input and ouput). 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.. *renderingPreference is an o*ptional list of rendering types, in order of preference. If	In • task identifier • properties • rendering preferences Out • task (`htt:tTaskInstanceData`)	No	(any state)	(no state transition)

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
getSubtasks	Returns all sub tasks of a task (created instances + not yet created sub task definitions)	In • task identifier Out • list of tasks (list of htt:tTask)	No	(any state)	(no state transition)
getSubtaskIdentifiers	Returns the identifiers of all already created sub tasks of a task	In • task identifier Out • list of task identifiers	No	(any state)	(no state transition)
hasSubtasks	Returns true if a task has at least one (already created or not yet created, but specified) sub task	In • task identifier Out • boolean	No	(any state)	(no state transition)
getParentTask	Returns the superior composite task of a sub task	In • task identifier Out • htt:tTask	No	(any state)	(no state transition)
getParentTaskIdentifier	Returns the task identifier of the superior composite task of a sub task	In • task identifier Out • task identifier	No	(any state)	(no state transition)
isSubtask	Returns true if a task is a sub task of a superior composite task	In • task identifier Out • boolean	No	(any state)	(no state transition)

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
instantiateSubTask	Creates an instantiateable subtask for the task from the definition of the task. The fault hta:illegalArgumentFault MUST be returned if the task does not have an instantiateable subtask of the given name. Returns the identifier for the created subtask.	In • task identifier • subtask name Out • task identifier	No	Reserved In Progress	(no state transition)
setTaskStartDeadlineExpression	Sets a deadline expression for the named start deadline of the task	In • task identifier • deadline name • deadline expression Out • void	Yes	Created Ready Reserved In Progress	(no state transition)
setTaskStartDurationExpression	Sets a duration expression for the named start deadline of the task	In • task identifier • deadline name • duration expression Out void	Yes	Created Ready Reserved In Progress	(no state transition)
setTaskCompletionDeadlineExpression	Sets a deadline expression for the named completion deadline of the task	In • task identifier • deadline name • deadline expression Out void	Yes	Created Ready Reserved In Progress	(no state transition)

Operation Name	Description	Parameters	Supports Batch Processing	Pre-State	Post-State
setTaskCompletionDurationExpression	Sets a duration expression for the named completion deadline of the task	In • task identifier • deadline name • duration expression Out void	Yes	Created Ready Reserved In Progress	(no state transition)

7.1.2 Simple Query Operations

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

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

Operation Name	Description	Parameters	Authorization
getMyTaskAbstracts	Retrieve the task abstracts. This operation is used to obtain the data required to display a task list. If no task type has been specified then the default value "ALL" MUST be used. If no generic human role has been specified then the default value "actualOwner" MUST be used. 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. If no status list has been specified then tasks in all valid states are returned. The where clause is	In • task type ("ALL" | "TASKS" | "NOTIFICATIONS") • generic human role • work queue • status list • where clause • order-by clause • created-on clause • maxTasks • taskIndexOffset Out • list of tasks (list of `htt:tTaskAbstract`)	Any

Operation Name	Description	Parameters	Authorization
	optional. If specified, it MUST reference exactly one column using the following operators: *equals* (“=”), *not equals* (“<>”), *less than* (“<”), *greater than* (“>”), *less than or equals* (“<=”), and *greater than or equals* (“>=”), e.g., “Task.Priority = 1”). The created-on clause is optional. The *where* clause is logically ANDed with the created-on clause, which MUST reference the column Task.CreatedTime with operators as described above. The combination of the two clauses enables simple but restricted paging in a task list client. 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.		
getMyTaskDetails	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. If no task type has been specified then	In • task type (“ALL” | “TASKS” | “NOTIFICATIONS”) • generic human role • work queue • status list	Any

Operation Name	Description	Parameters	Authorization
	the default value "ALL" MUST be used. If no generic human role has been specified then the default value "actualOwner" MUST be used. 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. If no status list has been specified then tasks in all valid states are returned. The where clause is optional. If specified, it MUST reference exactly one column using the following operators: *equals* ("="), *not equals* ("<>"), *less than* ("<"), *greater than* (">"), *less than or equals* ("<="), and *greater than or equals* (">="),e.g., "Task.Priority = 1". The created-on clause is optional. The *where* clause is logically ANDed with the created-on clause, which MUST reference the column Task.CreatedTime with operators as described above. The combination of the two clauses enables simple but restricted paging in the task list client. If maxTasks is specified, then the number of task details returned for this query	• where clause • created-on clause • maxTasks Out • list of tasks (list of `htt:tTaskDetails`)	

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

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

Simple Task View

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

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

Column Name	Type
SearchBy	`xsd:string`
Outcome	`xsd:string`

7.1.3 Advanced Query Operation

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

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

Operation Name	Description	Parameters
query	Retrieve task data. All clauses assume a (pseudo-) SQL syntax. If maxTasks is specified, then the number of task returned by the query MUST NOT exceed this limit. The taskIndexOffset can be used to perform multiple identical queries and iterate over result sets where the maxTasks size exceeds the query limit.	In • select clause • where clause • order-by clause • maxTasks • taskIndexOffset Out • task query result set (`htt:tTaskQueryResultSet`)

ResultSet Data Type

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

```
<xsd:element name="taskQueryResultSet" type="tTaskQueryResultSet" />
<xsd:complexType name="tTaskQueryResultSet">
  <xsd:sequence>
    <xsd:element name="row" type="tTaskQueryResultRow"
                 minOccurs="0" maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>
```

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

```
<xsd:complexType name="tTaskQueryResultRow">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="id" type="xsd:string"/>
    <xsd:element name="taskType" type="xsd:string"/>
    <xsd:element name="name" type="xsd:QName"/>
    <xsd:element name="status" type="tStatus"/>
    <xsd:element name="priority" type="htt:tPriority"/>
    <xsd:element name="taskInitiator"
                 type="htt:tUser"/>
    <xsd:element name="taskStakeholders"
                 type="htt:tOrganizationalEntity"/>
    <xsd:element name="potentialOwners"
                 type="htt:tOrganizationalEntity"/>
    <xsd:element name="businessAdministrators"
                 type="htt:tOrganizationalEntity"/>
    <xsd:element name="actualOwner" type="htt:tUser"/>
    <xsd:element name="notificationRecipients"
                 type="htt:tOrganizationalEntity"/>
    <xsd:element name="createdTime" type="xsd:dateTime"/>
    <xsd:element name="createdBy" type="xsd:string"/>
    <xsd:element name="lastModifiedTime" type="xsd:dateTime"/>
    <xsd:element name="lastModifiedBy" type="xsd:string"/>
    <xsd:element name="activationTime" type="xsd:dateTime"/>
    <xsd:element name="expirationTime" type="xsd:dateTime"/>
    <xsd:element name="isSkipable" type="xsd:boolean"/>
    <xsd:element name="hasPotentialOwners" type="xsd:boolean"/>
    <xsd:element name="startByTime" type="xsd:dateTime"/>
    <xsd:element name="completeByTime" type="xsd:dateTime"/>
    <xsd:element name="presentationName" type="tPresentationName"/>
    <xsd:element name="presentationSubject"
                 type="tPresentationSubject"/>
    <xsd:element name="renderingMethodName" type="xsd:QName"/>
    <xsd:element name="hasOutput" type="xsd:boolean"/>
    <xsd:element name="hasFault" type="xsd:boolean"/>
    <xsd:element name="hasAttachments" type="xsd:boolean"/>
    <xsd:element name="hasComments" type="xsd:boolean"/>
    <xsd:element name="escalated" type="xsd:boolean"/>
    <xsd:element name="parentTaskId" type="xsd:string"/>
    <xsd:element name="hasSubTasks" type="xsd:boolean"/>
    <xsd:element name="searchBy" type="xsd:string"/>
    <xsd:element name="outcome" type="xsd:string"/>
    <xsd:element name="taskOperations" type="tTaskOperations"/>
    <xsd:any namespace="##other" processContents="lax"/>
  </xsd:choice>
</xsd:complexType>
```

Complete Task View

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

Column Name	**Type**	**Constraints**
ID	`xsd:string`	
TaskType	Enumeration	Identifies the task type. The following values are allowed: • “TASK” for a human task • “NOTIFICATION” for notifications Note that notifications are simple tasks that do not block the progress of the caller,
Name	`xsd:QName`	
Status	Enumeration	For values see section 4.10 “Human Task Behavior and State Transitions”
Priority	`htt:tPriority`	
(GenericHumanRole)	`xsd:tUser` or `htt:tOrganizationalEntity`	
CreatedTime	`xsd:dateTime`	The time in UTC when the task has been created.
CreatedBy	`xsd:string`	
LastModifiedTime	`xsd:dateTime`	The time in UTC when the task has been last modified.
LastModifiedBy	`xsd:string`	
ActivationTime	`xsd:dateTime`	The time in UTC when the task has been activated.
ExpirationTime	`xsd:dateTime`	The time in UTC when the task will expire.
IsSkipable	`xsd:boolean`	
StartByTime	`xsd:dateTime`	The time in UTC when the task needs to be started. This time corresponds to the respective start deadline.
CompleteByTime	`xsd:dateTime`	The time in UTC when the task needs to be completed. This time corresponds to the respective end deadline.

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

7.1.4 Administrative Operations

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

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

7.1.5 Operation Authorizations

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

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

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

2725

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

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

7.2 XPath Extension Functions

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

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

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

Operation Name	**Description**	**Parameters**
getPotentialOwners	Returns the potential owners of the task. It MUST evaluate to an empty `htt:organizationalEntity` in case of an error. If the task name is not present the current task MUST be considered.	In • task name (optional) Out • potential owners (`htt:organizationalEntity`)
getActualOwner	Returns the actual owner of the task. It MUST evaluate to an empty `htt:user` in case there is no actual owner. If the task name is not present the current task MUST be considered.	In • task name (optional) Out • the actual owner (user id as `htt:user`)
getTaskInitiator	Returns the initiator of the task. It MUST evaluate to an empty `htt:user` in case there is no initiator. If the task name is not present the current task MUST be considered.	In • task name (optional) Out • the task initiator (user id as `htt:user`)
getTaskStakeholders	Returns the stakeholders of the task. It MUST evaluate to an empty `htt:organizationalEntity` in case of an error. If the task name is not present the current task MUST be considered.	In • task name (optional) Out • task stakeholders (`htt:organizationalEntity`)
getBusinessAdministrators	Returns the business administrators of the task. It MUST evaluate to an empty `htt:organizationalEntity` in case of an error. If the task name is not present the current task MUST be considered.	In • task name (optional) Out • business administrators (`htt:organizationalEntity`)
getExcludedOwners	Returns the excluded owners. It MUST evaluate to an empty `htt:organizationalE`	In • task name (optional) Out

Operation Name	Description	Parameters
	`ntity` in case of an error. If the task name is not present the current task MUST be considered.	• excluded owners (`htt:organizationalEntity`)
getTaskPriority	Returns the priority of the task. It MUST evaluate to “5” in case the priority is not explicitly set. If the task name is not present the current task MUST be considered.	In • task name (optional) Out • priority (htt:tPriority)
getInput	Returns the part of the task’s input message. If the task name is not present the current task MUST be considered.	In • part name • task name (optional) Out • input message part
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 • sub task name • part name • location path Out • node-set of output message element(s)
getSubtaskOutputs	Returns a node-set of simple-typed or complex-typed elements, constructed from the sub tasks’ output documents in a routing pattern. The string parameter contains a location path evaluated on each sub task’s output document. The individual node-sets are combined into the returned node-set. Only completed sub tasks of the current task MUST be considered	In • part name • location path Out • node-set of output message elements from sub tasks
getOutput	Returns the part of the task's output message. If the task name is not present the current task MUST be considered	In • part name • task name (optional) Out

Operation Name	Description	Parameters
		• output message part
getCountOfSubTasks	Returns the number of sub tasks of a task If the task name is not present the current task MUST be considered	In • task name (optional) Out • Number of the task sub-tasks. If the task doesn't have sub tasks then 0 is returned
getCountOfFinishedSubTasks	Returns the number of finished sub tasks of a task If the task name is not present the current task MUST be considered	In • task name (optional) Out • Number of the finished task sub-tasks. If the task doesn't have sub tasks then 0 is returned
getCountOfSubTasksInState	Returns the number of a task suubtasks that are in the specified state If the task name is not present the current task MUST be considered	In • state • task name (optional) Out • Number of the task sub tasks in the specified state. If the task doesn't have sub tasks then 0 is returned
getCountOfSubTasksWithOutcome	Returns the number of a task sub tasks that match the given outcome If the task name is not present the current task MUST be considered	In • outcome • task name (optional) Out • Number of the task sub tasks that match the specified outcome. If the task doesn't have sub tasks then 0 is returned
getLogicalPeopleGroup	Returns the value of a logical people group. In case of an error (e.g., when referencing a non existing logical people group) the `htt:organizationalEntity` MUST contain an empty user list. If the task name is not present the current task MUST be considered.	In • name of the logical people group • The optional parameters that follow MUST appear in pairs. Each pair is defined as: o the qualified name of a logical people group parameter o the value for the named logical people group parameter; it can be an

Operation Name	Description	Parameters
		XPath expression Out • the value of the logical people group (`htt:organizationalEntity`)
getOutcome	Returns the outcome of the task. It MUST evaluate to an empty string in case there is no outcome specified for the task. If the task name is not present the current task MUST be considered.	In • task name (optional) Out • the task outcome (`xsd:string`)
union	Constructs an organizationalEntity containing every user that occurs in **either set1 or set2**, eliminating duplicate users.	In • set1 (`htt:organizationalEntity` |`htt:user`) • set2 (`htt:organizationalEntity` |`htt:user`) Out • result (`htt:organizationalEntity`)
intersect	Constructs an organizationalEntity containing every user that occurs in **both set1 and set2**, eliminating duplicate users.	In • set1 (`htt:organizationalEntity` |`htt:user`) • set2 (`htt:organizationalEntity` |`htt:user`) Out • result (`htt:organizationalEntity`)
except	Constructs an organizationalEntity containing every user that occurs in **set1 but not in set2**. Note: This function is required to allow enforcing the separation of duties	In • set1 (`htt:organizationalEntity` |`htt:user`) • set2 (`htt:organizationalEntity` |`htt:user`) Out

Operation Name	Description	Parameters
	(“4-eyes principle”).	• result (`htt:organizationalEntity`)

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

Operation Name	Description	Parameters
concat	Returns the concatenation of all string nodes - returns an empty string for an empty node-set	In • node-set of string nodes
concatWithDelimiter	Returns the concatenation of all string nodes, separated by the specified delimiter string - returns an empty string for an empty node-set	In • node-set of string nodes • delimiter string
leastFrequentOccurence	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 • node-set of string nodes
mostFrequentOccurence	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 • node-set of string nodes
voteOnString	Returns the most frequently occurring string value if its occurrence is above the specified percentage and there is no tie, or an empty string otherwise (including an empty node-set)	In • node-set of string nodes • percentage
and	Returns the conjunction of all boolean nodes - returns false for an empty node-set	In • node-set of boolean nodes
or	Returns the disjunction of all boolean nodes - returns false for an empty node-set	In • node-set of boolean nodes

Operation Name	Description	Parameters
vote	Returns the most frequently occurring boolean value if its occurrence is above the specified percentage, or false otherwise (including an empty node-set)	In • node-set of boolean nodes • percentage
avg	Returns the average value of all number nodes - returns NaN for an empty node-set	In • node-set of number nodes
max	Returns the maximum value of all number nodes - returns NaN for an empty node-set	In • node-set of number nodes
min	Returns the minimum value of all number nodes - returns NaN for an empty node-set	In • node-set of number nodes
sum	Returns the sum value of all number nodes - returns 0 for an empty node-set	In • node-set of number nodes

8 Interoperable Protocol for Advanced Interaction with Human Tasks

Previous sections describe how to define standard invokable 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.

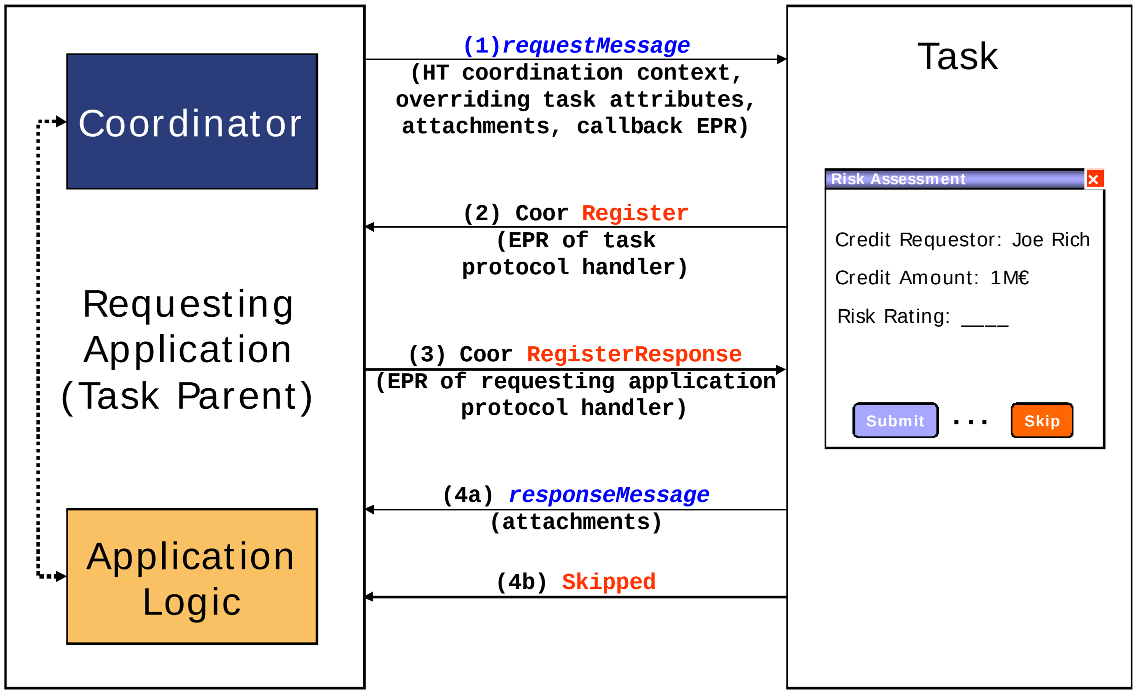

Coordinator
Requesting Application (Task Parent)
Application Logic
Task
(1)requestMessage
(HT coordination context, overriding task attributes, attachments, callback EPR)
(2) Coor Register
(EPR of task protocol handler)
(3) Coor RegisterResponse
(EPR of requesting application protocol handler)
(4a) responseMessage
(attachments)
(4b) Skipped
Risk Assessment
Credit Requestor: Joe Rich
Credit Amount: 1M€
Risk Rating: ____
Submit
...
Skip

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

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

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

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

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

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

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

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

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

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

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

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

8.1 Human Task Coordination Protocol Messages

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

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

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

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

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

8.2 Protocol Messages

All WS-HumanTask protocol messages have the following type:

```
<xsd:complexType name="tProtocolMsgType">
  <xsd:sequence>
    <xsd:any namespace="##other" processContents="lax"
             minOccurs="0" maxOccurs="unbounded" />
  </xsd:sequence>
  <xsd:anyAttribute namespace="##other" processContents="lax" />
</xsd:complexType>
```

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

8.2.1 Protocol Messages Received by a Task Parent

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

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

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

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

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

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

8.2.2 Protocol Messages Received by a Task

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

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

8.3 WSDL of the Protocol Endpoints

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

8.3.1 Protocol Endpoint of the Task Parent

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

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

```
  <wsdl:operation name="skippedOperation">
    <wsdl:input message="htcp:skipped" />
  </wsdl:operation>
  <wsdl:operation name="faultOperation">
    <wsdl:input message="htcp:fault" />
  </wsdl:operation>
</wsdl:portType>
```

8.3.2 Protocol Endpoint of the Task

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

```
<wsdl:portType name="humanTaskParticipantPortType">
  <wsdl:operation name="exitOperation">
    <wsdl:input message="htcp:exit" />
  </wsdl:operation>
</wsdl:portType>
```

8.4 Providing Human Task Context

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

8.4.1 SOAP Binding of Human Task Context

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

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
            xmlns:htc="http://docs.oasis-open.org/ns/bpel4people/ws-
humantask/context/200803">
 <S:Header>
   <htc:humanTaskRequestContext>
     <htc:priority>...</htc:priority>?
     <htc:attachments>...</htc:attachments>?
     <htc:peopleAssignments>...</htc:peopleAssignments>?
     <htc:isSkipable>...</htc:isSkipable>?
     <htc:activationDeferralTime>...</htc:activationDeferralTime>?
     <htc:expirationTime>...</htc:expirationTime>?
       ... extension elements ...
    </htc:humanTaskRequestContext>
  </S:Header>
  <S:Body>
    ...
  </S:Body>
</S:Envelope>
```

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

```
            xmlns:htc="http://docs.oasis-open.org/ns/bpel4people/ws-
humantask/context/200803">
 <S:Header>
   <htc:humanTaskResponseContext>
     <htc:priority>...</htc:priority>?
     <htc:attachments>...</htc:attachments>?
     <htc:actualOwner>...</htc:actualOwner>?
     <htc:actualPeopleAssignments>...</htc:actualPeopleAssignments>?
     <htc:outcome>...</htc:outcome>?
       ... extension elements ...
    </htc:humanTaskResponseContext>
  </S:Header>
  <S:Body>
    ...
  </S:Body>
</S:Envelope>
```

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

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
            xmlns:htc="http://docs.oasis-open.org/ns/bpel4people/ws-
humantask/context/200803">
  <S:Header>
    <htc:humanTaskRequestContext>
      <htc:priority>0</htc:priority>
      <htc:peopleAssignments>
        <htc:potentialOwners>
          <htt:organizationalEntity>
            <htt:user>Alan</htt:user>
            <htt:user>Dieter</htt:user>
            <htt:user>Frank</htt:user>
            <htt:user>Gerhard</htt:user>
            <htt:user>Ivana</htt:user>
            <htt:user>Karsten</htt:user>
            <htt:user>Matthias</htt:user>
            <htt:user>Patrick</htt:user>
          </htt:organizationalEntity>
        </htc:potentialOwners>
      </htc:peopleAssignments>
    </htc:humanTaskRequestContext>
  </S:Header>
  <S:Body>...</S:Body>
</S:Envelope>
```

8.4.2 Overriding Task Definition People Assignments

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

8.5 Human Task Policy Assertion

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

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

```
<htp:HumanTaskAssertion wsp:Optional="true"? ...>
  ...
</htp:HumanTaskAssertion>
```

/htp:HumanTaskAssertion

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

/htp:HumanTaskAssertion/@wsp:Optional="true"

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

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

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

9 Task Parent Interactions with Lean Tasks

9.1 Operations for Task Parent Applications

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

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

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

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

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

9.2 Lean Task Interactions

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

- `registerLeanTaskDefinition` API for registration
- `unregisterLeanTaskDefinition` API for retraction
- `listLeanTaskDefinitions` API for enumeration
- `createLeanTask` and `createLeanTaskAsync` APIs for creation

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

- `createLeanTaskAsyncCallback`

9.2.1 Register a Lean Task Definition

```
<xsd:element name="registerLeanTaskDefinition">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="taskDefinition" type="htd:tLeanTask" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="registerLeanTaskDefinitionResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="taskName" type="xsd:NCName" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

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

9.2.2 Unregister a Lean Task Definition

```
  <xsd:element name="unregisterLeanTaskDefinition">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="taskName" type="xsd:NCName" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="unregisterLeanTaskDefinitionResponse">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="taskName" type="xsd:NCName" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
```

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

9.2.3 List Lean Task Definitions

```
  <xsd:element name="listLeanTaskDefinitions">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:annotation>
          <xsd:documentation>Empty message</xsd:documentation>
        </xsd:annotation>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="listLeanTaskDefinitionsResponse">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="leanTaskDefinitions">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="leanTaskDefinition" type="htd:tLeanTask"
minOccurs="0" maxOccurs="unbounded" />
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
```

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

9.2.4 Create a Lean Task

```
<xsd:element name="CreateLeanTask">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="inputMessage">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:any processContents="lax" namespace="##any" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="taskDefinition" type="htd:tLeanTask" minOccurs="0"/>
      <xsd:element name="taskName" type="xsd:NCName" minOccurs="0" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="CreateLeanTaskResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="outputMessage">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:any processContents="lax" namespace="##any" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

```

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

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

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

Alternatively, instead of invoking the long-running request-response operation defined above, an interaction style using an asynchronous callback operation can be used. In this case, the WS-HumanTask Parent invokes the following htlt:createLeanTaskAsync operation and, as described in section 10, passes a WS-Addressing endpoint reference (EPR) in order to provide a callback address for delivering the lean task's output.

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

```
<xsd:element name="createLeanTaskAsync">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="inputMessage">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:any processContents="lax" namespace="##any" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="taskDefinition" type="htd:tLeanTask" minOccurs="0"/>
      <xsd:element name="taskName" type="xsd:NCName" minOccurs="0" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="createLeanTaskAsyncResponse">
  <xsd:complexType>
    <xsd:sequence/>
  </xsd:complexType>
</xsd:element>
```

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

```
<xsd:element name="createLeanTaskAsyncCallback">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="outputMessage">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:any processContents="lax" namespace="##any" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

9.2.5 Endpoints for Lean Task Operations

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

```
<wsdl:portType name="leanTaskOperations">

  <wsdl:operation name="registerLeanTaskDefinition">
    <wsdl:input message="registerLeanTaskDefinition"/>
    <wsdl:output message="registerLeanTaskDefinitionResponse"/>
    <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
    <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
  </wsdl:operation>

  <wsdl:operation name="unregisterLeanTaskDefinition">
    <wsdl:input message="unregisterLeanTaskDefinition"/>
    <wsdl:output message="unregisterLeanTaskDefinitionResponse"/>
    <wsdl:fault name="illegalArgumentFault" message="illegalArgumentFault"/>
    <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
  </wsdl:operation>

  <wsdl:operation name="listLeanTaskDefinitions">
    <wsdl:input message="listLeanTaskDefinitions"/>
    <wsdl:output message="listLeanTaskDefinitionsResponse"/>
    <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
  </wsdl:operation>

  <wsdl:operation name="createLeanTask">
    <wsdl:input message="createLeanTask"/>
    <wsdl:output message="createLeanTaskResponse"/>
    <wsdl:fault name="illegalArgumentFault" message="illegalArgumentFault"/>
    <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
  </wsdl:operation>

  <wsdl:operation name="createLeanTaskAsync">
    <wsdl:input message="createLeanTaskAsync"/>
    <wsdl:output message="createLeanTaskAsyncResponse"/>
    <wsdl:fault name="illegalArgumentFault" message="illegalArgumentFault"/>
    <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
  </wsdl:operation>

</wsdl:portType>
```

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

```
<wsdl:portType name="leanTaskCallbackOperations">

  <wsdl:operation name="createLeanTaskAsyncCallback">
    <wsdl:input message="createLeanTaskAsyncCallback"/>
  </wsdl:operation>

</wsdl:portType>
```

10 Providing Callback Information for Human Tasks

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

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

10.1 EPR Information Model Extension

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

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

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

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

    This property contains the name of a WSDL operation.

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

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

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

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

10.2 XML Infoset Representation

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

```
<wsa:EndpointReference>
  <wsa:Address>xsd:anyURI</wsa:Address>
  <wsa:ReferenceParameters>xsd:any*</wsa:ReferenceParameters>?
  <wsa:Metadata>
    <htcp:responseAction>xsd:anyURI</htcp:responseAction>?
    <htcp:responseOperation>xsd:NCName</htcp:responseOperation>?
  </wsa:Metadata>
```

```
</wsa:EndpointReference>
```

/wsa:EndpointReference/wsa:Metadata

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

/wsa:EndpointReference/wsa:Metadata/htcp:responseAction

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

/wsa:EndpointReference/wsa:Metadata/htcp:responseOperation

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

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

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

```
<wsa:EndpointReference
  xmlns:wsa="http://www.w3.org/2005/08/addressing">

  <wsa:Address>http://example.com/LoanApproval/loan?ID=42</wsa:Address>

  <wsa:Metadata>
    <htcp:responseAction>
      http://example.com/LoanApproval/approvalResponse
    </htcp:responseAction>
  </wsa:Metadata>

</wsa:EndpointReference>
```

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

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

```
<wsa:EndpointReference
  xmlns:wsa="http://www.w3.org/2005/08/addressing">

  <wsa:Address>http://example.com/LoanApproval/loan</wsa:Address>

  <wsa:ReferenceParameters>
    <xmp:MyInstanceID>42</xmp:MyInstanceID>
  </wsa:ReferenceParameters>

  <wsa:Metadata>

    <wsdl:definitions ...>

      <wsdl:portType name="LoanApprovalPT">
        <wsdl:operation name="approvalResponse">...</wsdl:operation>
        ...
      </wsdl:portType>

      <wsdl:binding name="LoanApprovalSoap" type="LoanApprovalPT">
        ...
      </wsdl:binding>

      <wsdl:service name="LoanApprovalService">
        <wsdl:port name="LA" binding="LoanApprovalSoap">
          <soap:address
            location="http://example.com/LoanApproval/loan" />
        </wsdl:port>
        ...
      </wsdl:service>

    </wsdl:definitions>

    <htcp:responseOperation>approvalResponse</htcp:responseOperation>

  </wsa:Metadata>

</wsa:EndpointReference>
```

10.3 Message Addressing Properties

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

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

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

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

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

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

10.4 SOAP Binding

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

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

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="http://www.w3.org/2005/08/addressing"
  xmlns:htcp="http://docs.oasis-open.org/ns/bpel4people/ws-
humantask/protocol/200803">

  <S:Header>
    <wsa:ReplyTo>
      <wsa:Address>http://example.com/LoanApproval/loan</wsa:Address>
      <wsa:ReferenceParameters>
        <xmp:MyInstanceID>42</xmp:MyInstanceID>
      </wsa:ReferenceParameters>
      <wsa:Metadata>
        <htcp:responseAction>
          http://example.com/LoanApproval/approvalResponse
        </htcp:responseAction>
      </wsa:Metadata>
    </wsa:ReplyTo>
  </S:Header>

  <S:Body>...</S:Body>
```

```
</S:Envelope>
```

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

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="http://www.w3.org/2005/08/addressing">
  <S:Header>
    <wsa:To>
      <wsa:Address>http://example.com/LoanApproval/loan</wsa:Address>
    </wsa:To>
    <wsa:Action>
      http://example.com/LoanApproval/approvalResponse
    </wsa:Action>
    <xmp:MyInstanceID wsa:IsReferenceParameter='true'>
      42
    </xmp:MyInstanceID>
  </S:Header>
  <S:Body>...</S:Body>
</S:Envelope>
```

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

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="http://www.w3.org/2005/08/addressing"
  xmlns:htcp="http://docs.oasis-open.org/ns/bpel4people/ws-
humantask/protocol/200803">
  <S:Header>
    <wsa:ReplyTo>

      <wsa:Address>http://example.com/LoanApproval/loan</wsa:Address>

      <wsa:ReferenceParameters>
        <xmp:MyInstanceID>42</xmp:MyInstanceID>
      </wsa:ReferenceParameters>

      <wsa:Metadata>

        <wsdl:definitions
          targetNamespace="http://example.com/loanApproval"
          xmlns:wsdl="..." xmlns:soap="...">

          <wsdl:portType name="LoanApprovalPT">
            <wsdl:operation name="approvalResponse">
              <wsdl:input name="approvalInput" ... />
            </wsdl:operation>
            ...
          </wsdl:portType>

          <wsdl:binding name="LoanApprovalSoap"
            type="LoanApprovalPT">
            ...
          </wsdl:binding>

          <wsdl:service name="LoanApprovalService">
            <wsdl:port name="LA" binding="LoanApprovalSoap">
              <soap:address
                location="http://example.com/LoanApproval/loan" />
            </wsdl:port>
            ...
          </wsdl:service>
        </wsdl:definitions>

        <htcp:responseOperation>
          approvalResponse
        </htcp:responseOperation>

      </wsa:Metadata>
    </wsa:ReplyTo>

  </S:Header>
  <S:Body>...</S:Body>
</S:Envelope>
```

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

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="http://www.w3.org/2005/08/addressing"
  xmlns:htd="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803">
  <S:Header>
    <wsa:To>http://example.com/LoanApproval/loan</wsa:To>
    <wsa:Action>
      http://example.com/loanApproval/...
      ...LoanApprovalPT/approvalResponse/ApprovalInput
    </wsa:Action>
    <xmp:MyInstanceID wsa:IsReferenceParameter='true'>
      42
    </xmp:MyInstanceID>
  </S:Header>
  <S:Body>...</S:Body>
</S:Envelope>
```

11 Security Considerations

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

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

12 Conformance

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

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

A. Portability and Interoperability Considerations

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

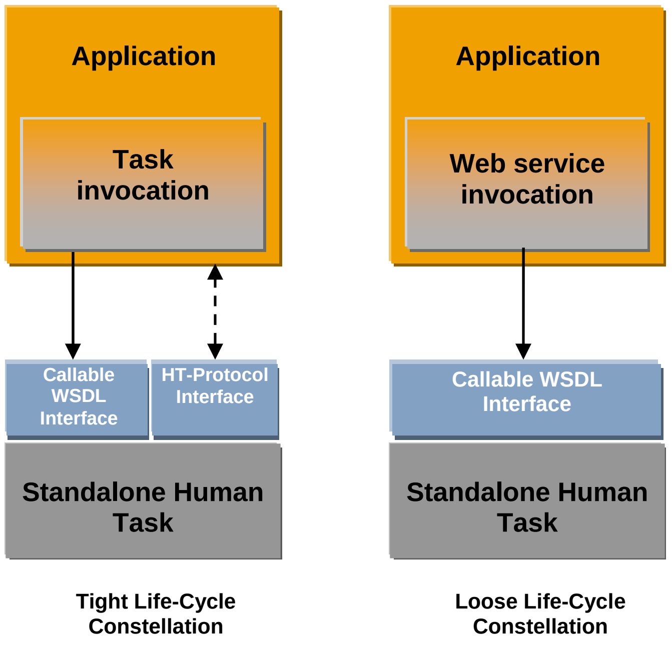
- Portability - The ability to take human tasks and notifications created in one vendor's environment and use them in another vendor's environment.
- Interoperability - The capability for multiple components (task infrastructure, task list clients and applications or processes with human interactions)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

Portability requires support of WS-HumanTask artifacts.

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

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 invocating application are coupled with each other. This is shown in the figure below:

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



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.

B. WS-HumanTask Language Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Copyright (c) OASIS Open 2009. All Rights Reserved.
-->
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803"
  targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
humantask/200803"
  elementFormDefault="qualified" blockDefault="#all">

  <xsd:annotation>
    <xsd:documentation>
      XML Schema for WS-HumanTask 1.1 - WS-HumanTask Task Definition Language
    </xsd:documentation>
  </xsd:annotation>

  <!-- other namespaces -->
  <xsd:import namespace="http://www.w3.org/XML/1998/namespace"
    schemaLocation="http://www.w3.org/2001/xml.xsd" />

  <!-- base types for extensible elements -->
  <xsd:complexType name="tExtensibleElements">
    <xsd:sequence>
      <xsd:element name="documentation" type="tDocumentation" minOccurs="0"
maxOccurs="unbounded" />
      <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
  </xsd:complexType>

  <xsd:complexType name="tDocumentation" mixed="true">
    <xsd:sequence>
      <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:attribute ref="xml:lang" />
  </xsd:complexType>

  <xsd:complexType name="tExtensibleMixedContentElements"
    mixed="true">
    <xsd:sequence>
      <xsd:element name="documentation" type="tDocumentation" minOccurs="0"
maxOccurs="unbounded" />
      <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
  </xsd:complexType>

  <!-- human interactions definition -->
  <xsd:element name="humanInteractions" type="tHumanInteractions" />
  <xsd:complexType name="tHumanInteractions">
```

```
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element name="extensions" type="tExtensions" minOccurs="0" />
          <xsd:element name="import" type="tImport" minOccurs="0"
maxOccurs="unbounded" />
          <xsd:element name="logicalPeopleGroups" type="tLogicalPeopleGroups"
minOccurs="0" />
          <xsd:element name="tasks" type="tTasks" minOccurs="0" />
          <xsd:element name="notifications" type="tNotifications"
minOccurs="0" />
        </xsd:sequence>
        <xsd:attribute name="targetNamespace" type="xsd:anyURI"
use="required" />
        <xsd:attribute name="queryLanguage" type="xsd:anyURI" />
        <xsd:attribute name="expressionLanguage" type="xsd:anyURI" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tExtensions">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element name="extension" type="tExtension"
maxOccurs="unbounded" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tExtension">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:attribute name="namespace" type="xsd:anyURI" use="required" />
        <xsd:attribute name="mustUnderstand" type="tBoolean" use="required"
/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="import" type="tImport" />
  <xsd:complexType name="tImport">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:attribute name="namespace" type="xsd:anyURI" use="optional" />
        <xsd:attribute name="location" type="xsd:anyURI" use="optional" />
        <xsd:attribute name="importType" type="xsd:anyURI" use="required" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="logicalPeopleGroups" type="tLogicalPeopleGroups" />
  <xsd:complexType name="tLogicalPeopleGroups">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
```

```
          <xsd:element name="logicalPeopleGroup" type="tLogicalPeopleGroup"
maxOccurs="unbounded" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tLogicalPeopleGroup">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element name="parameter" type="tParameter" minOccurs="0"
maxOccurs="unbounded" />
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:NCName" use="required" />
        <xsd:attribute name="reference" type="xsd:NCName" use="optional" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <!-- generic human roles used in tasks and notifications -->
  <xsd:element name="genericHumanRole" type="tGenericHumanRoleAssignmentBase"
abstract="true" block=""/>

  <xsd:element name="potentialOwners" type="tPotentialOwnerAssignment"
substitutionGroup="genericHumanRole"/>
  <xsd:element name="excludedOwners" type="tGenericHumanRoleAssignment"
substitutionGroup="genericHumanRole"/>
  <xsd:element name="taskInitiator" type="tGenericHumanRoleAssignment"
substitutionGroup="genericHumanRole"/>
  <xsd:element name="taskStakeholders" type="tGenericHumanRoleAssignment"
substitutionGroup="genericHumanRole"/>
  <xsd:element name="businessAdministrators"
type="tGenericHumanRoleAssignment" substitutionGroup="genericHumanRole"/>
  <xsd:element name="recipients" type="tGenericHumanRoleAssignment"
substitutionGroup="genericHumanRole"/>

  <xsd:complexType name="tGenericHumanRoleAssignmentBase" block="">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements"/>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tGenericHumanRoleAssignment">
    <xsd:complexContent>
      <xsd:extension base="tGenericHumanRoleAssignmentBase">
        <xsd:sequence>
          <xsd:element name="from" type="tFrom" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tPotentialOwnerAssignment">
    <xsd:complexContent>
      <xsd:extension base="tGenericHumanRoleAssignmentBase">
        <xsd:choice>
          <xsd:element name="from" type="tFrom" />
          <xsd:element name="parallel" type="tParallel" />
          <xsd:element name="sequence" type="tSequence" />
        </xsd:choice>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <!-- routing patterns -->
  <xsd:complexType name="tParallel">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element name="completionBehavior" type="tCompletionBehavior"
minOccurs="0" />
          <xsd:element name="from" type="tFrom" minOccurs="0"
maxOccurs="unbounded" />
          <xsd:choice minOccurs="0" maxOccurs="unbounded">
            <xsd:element name="parallel" type="tParallel" />
            <xsd:element name="sequence" type="tSequence" />
          </xsd:choice>
        </xsd:sequence>
        <xsd:attribute name="type" type="tRoutingPatternType" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tSequence">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element name="completionBehavior" type="tCompletionBehavior"
/>
          <xsd:element name="from" type="tFrom" minOccurs="0"
maxOccurs="unbounded" />
          <xsd:choice minOccurs="0" maxOccurs="unbounded">
            <xsd:element name="parallel" type="tParallel" />
            <xsd:element name="sequence" type="tSequence" />
          </xsd:choice>
        </xsd:sequence>
        <xsd:attribute name="type" type="tRoutingPatternType" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:simpleType name="tRoutingPatternType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="all" />
      <xsd:enumeration value="single" />
    </xsd:restriction>
  </xsd:simpleType>

  <!-- completion behavior -->
  <xsd:complexType name="tCompletionBehavior">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element name="completion" type="tCompletion" minOccurs="0"
maxOccurs="unbounded" />
```

```
          <xsd:element name="defaultCompletion" type="tDefaultCompletion"
minOccurs="0" />
        </xsd:sequence>
        <xsd:attribute name="completionAction" type="tPattern" use="optional"
default="automatic" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tCompletion">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element name="condition" type="tBoolean-expr" />
          <xsd:element name="result" type="tResult" minOccurs="0" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tDefaultCompletion">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element name="result" type="tResult" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <!-- result construction -->
  <xsd:complexType name="tResult">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:choice maxOccurs="unbounded">
          <xsd:element name="aggregate" type="tAggregate" />
          <xsd:element name="copy" type="tCopy" />
        </xsd:choice>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tAggregate">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:attribute name="part" type="xsd:NCName" use="optional" />
        <xsd:attribute name="location" type="xsd:string" use="optional" />
        <xsd:attribute name="condition" type="xsd:string" />
        <xsd:attribute name="function" type="xsd:string" use="required" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tCopy">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element name="from" type="tExpression" />
          <xsd:element name="to" type="tQuery" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <!-- human tasks -->
  <xsd:element name="tasks" type="tTasks" />
  <xsd:complexType name="tTasks">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element name="task" type="tTask" maxOccurs="unbounded" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tTaskBase" abstract="true">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element name="interface" type="tTaskInterface" minOccurs="0"
/>
          <xsd:element name="messageSchema" type="tMessageSchema"
minOccurs="0" />
          <xsd:element name="priority" type="tPriority-expr" minOccurs="0" />
          <xsd:element name="peopleAssignments" type="tPeopleAssignments"
minOccurs="0" />
          <xsd:element name="completionBehavior" type="tCompletionBehavior"
minOccurs="0" />
          <xsd:element name="delegation" type="tDelegation" minOccurs="0" />
          <xsd:element name="presentationElements"
type="tPresentationElements" minOccurs="0" />
          <xsd:element name="outcome" type="tQuery" minOccurs="0" />
          <xsd:element name="searchBy" type="tExpression" minOccurs="0" />
          <xsd:element name="renderings" type="tRenderings" minOccurs="0" />
          <xsd:element name="deadlines" type="tDeadlines" minOccurs="0" />
          <xsd:element name="composition" type="tComposition" minOccurs="0"
/>
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:NCName" use="required" />
        <xsd:attribute name="actualOwnerRequired" type="tBoolean"
use="optional" default="yes" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="task" type="tTask" />
  <xsd:complexType name="tTask">
    <xsd:complexContent>
      <xsd:restriction base="tTaskBase">
        <xsd:sequence>
          <xsd:element name="documentation" type="tDocumentation"
minOccurs="0" maxOccurs="unbounded" />
          <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded" />
          <xsd:element name="interface" type="tTaskInterface" />
          <xsd:element name="messageSchema" type="tMessageSchema"
minOccurs="0" maxOccurs="0" />
          <xsd:element name="priority" type="tPriority-expr" minOccurs="0" />
          <xsd:element name="peopleAssignments" type="tPeopleAssignments"
minOccurs="0" />
          <xsd:element name="completionBehavior" type="tCompletionBehavior"
minOccurs="0" />
         <xsd:element name="delegation" type="tDelegation" minOccurs="0" />
          <xsd:element name="presentationElements"
type="tPresentationElements" minOccurs="0" />
          <xsd:element name="outcome" type="tQuery" minOccurs="0" />
          <xsd:element name="searchBy" type="tExpression" minOccurs="0" />
          <xsd:element name="renderings" type="tRenderings" minOccurs="0" />
          <xsd:element name="deadlines" type="tDeadlines" minOccurs="0" />
          <xsd:element name="composition" type="tComposition" minOccurs="0"
/>
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:NCName" use="required" />
        <xsd:attribute name="actualOwnerRequired" type="tBoolean"
use="optional" default="yes" />
        <xsd:anyAttribute namespace="##other" processContents="lax" />
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tTaskInterface">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:attribute name="portType" type="xsd:QName" use="required" />
        <xsd:attribute name="operation" type="xsd:NCName" use="required" />
        <xsd:attribute name="responsePortType" type="xsd:QName"
use="optional" />
        <xsd:attribute name="responseOperation" type="xsd:NCName"
use="optional" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <!-- presentation elements -->
  <xsd:complexType name="tPresentationElements">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element name="name" type="tText" minOccurs="0"
maxOccurs="unbounded" />
          <xsd:element name="presentationParameters"
type="tPresentationParameters" minOccurs="0" />
          <xsd:element name="subject" type="tText" minOccurs="0"
maxOccurs="unbounded" />
          <xsd:element name="description" type="tDescription" minOccurs="0"
maxOccurs="unbounded" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tPresentationParameters">
    <xsd:complexContent>
```

```
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element name="presentationParameter" 
type="tPresentationParameter" maxOccurs="unbounded" />
        </xsd:sequence>
        <xsd:attribute name="expressionLanguage" type="xsd:anyURI" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tPresentationParameter">
    <xsd:complexContent>
      <xsd:extension base="tParameter" />
    </xsd:complexContent>
  </xsd:complexType>

  <!-- elements for rendering tasks -->
  <xsd:complexType name="tRenderings">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element name="rendering" type="tRendering" 
maxOccurs="unbounded" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tRendering">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:attribute name="type" type="xsd:QName" use="required" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <!-- elements for people assignment -->
  <xsd:element name="peopleAssignments" type="tPeopleAssignments" />
  <xsd:complexType name="tPeopleAssignments">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element ref="genericHumanRole" minOccurs="0" 
maxOccurs="unbounded" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <!-- elements for handling timeouts and escalation -->
  <xsd:complexType name="tDeadlines">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element name="startDeadline" type="tDeadline" minOccurs="0" 
maxOccurs="unbounded" />
          <xsd:element name="completionDeadline" type="tDeadline" 
minOccurs="0" maxOccurs="unbounded" />
```

```
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tDeadline">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:choice>
            <xsd:element name="for" type="tDuration-expr" />
            <xsd:element name="until" type="tDeadline-expr" />
          </xsd:choice>
          <xsd:element name="escalation" type="tEscalation" minOccurs="0"
maxOccurs="unbounded" />
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:NCName" use="required"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tEscalation">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element name="condition" type="tBoolean-expr" minOccurs="0" />
          <xsd:element name="toParts" type="tToParts" minOccurs="0" />
          <xsd:choice>
            <xsd:element name="notification" type="tNotification" />
            <xsd:element name="localNotification" type="tLocalNotification"
/>
            <xsd:element name="reassignment" type="tReassignment" />
          </xsd:choice>
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:NCName" use="required" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tLocalNotification">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:choice>
          <xsd:sequence>
            <xsd:element name="priority" type="tPriority-expr" minOccurs="0"
/>
            <xsd:element name="peopleAssignments" type="tPeopleAssignments"
minOccurs="0" />
          </xsd:sequence>
        </xsd:choice>
        <xsd:attribute name="reference" type="xsd:QName" use="required" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tReassignment">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
```

```
        <xsd:sequence>
          <xsd:element ref="potentialOwners" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tToParts">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element name="toPart" type="tToPart" maxOccurs="unbounded" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tToPart" mixed="true">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleMixedContentElements">
        <xsd:attribute name="name" type="xsd:NCName" use="required" />
        <xsd:attribute name="expressionLanguage" type="xsd:anyURI" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <!-- task delegation -->
  <xsd:complexType name="tDelegation">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element name="from" type="tFrom" minOccurs="0" />
        </xsd:sequence>
        <xsd:attribute name="potentialDelegatees" type="tPotentialDelegatees"
use="required" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:simpleType name="tPotentialDelegatees">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="anybody" />
      <xsd:enumeration value="nobody" />
      <xsd:enumeration value="potentialOwners" />
      <xsd:enumeration value="other" />
    </xsd:restriction>
  </xsd:simpleType>

  <!-- composite tasks -->
  <xsd:complexType name="tComposition">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element name="subtask" type="tSubtask" maxOccurs="unbounded"
/>
        </xsd:sequence>
        <xsd:attribute name="type" type="tCompositionType" use="optional"
default="sequential" />
```

```
        <xsd:attribute name="instantiationPattern" type="tPattern"
use="optional" default="manual" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:simpleType name="tCompositionType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="sequential" />
      <xsd:enumeration value="parallel" />
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="tPattern">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="manual" />
      <xsd:enumeration value="automatic" />
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="tSubtask">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:choice>
          <xsd:element name="task" type="tTask"/>
          <xsd:element name="localTask" type="tLocalTask" />
        </xsd:choice>
        <xsd:attribute name="name" type="xsd:NCName" use="required" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tLocalTask">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element name="priority" type="tPriority-expr" minOccurs="0" />
          <xsd:element name="peopleAssignments" type="tPeopleAssignments"
minOccurs="0" />
        </xsd:sequence>
        <xsd:attribute name="reference" type="xsd:QName" use="required" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <!-- lean tasks -->
  <xsd:element name="leanTask" type="tLeanTask"/>
  <xsd:complexType name="tLeanTask">
    <xsd:complexContent>
      <xsd:restriction base="tTaskBase">
        <xsd:sequence>
          <xsd:element name="documentation" type="tDocumentation"
minOccurs="0" maxOccurs="unbounded" />
          <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded" />
          <xsd:element name="interface" type="tTaskInterface" minOccurs="0"
maxOccurs="0" />
          <xsd:element name="messageSchema" type="tMessageSchema" />
          <xsd:element name="priority" type="tPriority-expr" minOccurs="0" />
          <xsd:element name="peopleAssignments" type="tPeopleAssignments"
minOccurs="0" />
          <xsd:element name="delegation" type="tDelegation" minOccurs="0" />
          <xsd:element name="presentationElements"
type="tPresentationElements" minOccurs="0" />
          <xsd:element name="outcome" type="tQuery" minOccurs="0" />
          <xsd:element name="searchBy" type="tExpression" minOccurs="0" />
          <xsd:element name="renderings" type="tRenderings" minOccurs="0" />
          <xsd:element name="deadlines" type="tDeadlines" minOccurs="0" />
          <xsd:element name="composition" type="tComposition" minOccurs="0"
maxOccurs="0" />
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:NCName" use="required" />
        <xsd:attribute name="actualOwnerRequired" type="tBoolean"
use="optional" default="yes" />
        <xsd:anyAttribute namespace="##other" processContents="lax" />
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tMessageSchema">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
           <xsd:element name="messageField" type="tMessageField"
minOccurs="0" maxOccurs="unbounded" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tMessageField">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
           <xsd:element name="messageDisplay" type="tMessageDisplay"
maxOccurs="unbounded" />
           <xsd:element name="messageChoice" type="tMessageChoice"
minOccurs="0" maxOccurs="unbounded" />
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:NCName" />
        <xsd:attribute name="type" type="xsd:QName" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tMessageChoice">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
           <xsd:element name="messageDisplay" type="tMessageDisplay"
maxOccurs="unbounded" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tMessageDisplay">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:attribute ref="xml:lang" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <!-- notifications -->
  <xsd:element name="notifications" type="tNotifications" />
  <xsd:complexType name="tNotifications">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element name="notification" type="tNotification"
maxOccurs="unbounded" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="notification" type="tNotification" />
  <xsd:complexType name="tNotification">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:sequence>
          <xsd:element name="interface" type="tNotificationInterface" />
          <xsd:element name="priority" type="tPriority-expr" minOccurs="0" />
          <xsd:element name="peopleAssignments" type="tPeopleAssignments" />
          <xsd:element name="presentationElements"
type="tPresentationElements" />
          <xsd:element name="renderings" type="tRenderings" minOccurs="0" />
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:NCName" use="required" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tNotificationInterface">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
        <xsd:attribute name="portType" type="xsd:QName" use="required" />
        <xsd:attribute name="operation" type="xsd:NCName" use="required" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <!-- miscellaneous helper types -->
  <xsd:complexType name="tText" mixed="true">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleMixedContentElements">
        <xsd:attribute ref="xml:lang" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tDescription" mixed="true">
    <xsd:complexContent>
```

```
      <xsd:extension base="tExtensibleMixedContentElements">
        <xsd:attribute ref="xml:lang" />
        <xsd:attribute name="contentType" type="xsd:string" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tFrom" mixed="true">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleMixedContentElements">
        <xsd:sequence>
          <xsd:choice>
            <xsd:element name="argument" type="tArgument" minOccurs="0"
maxOccurs="unbounded"/>
            <xsd:element name="literal" type="tLiteral" minOccurs="0" />
          </xsd:choice>
        </xsd:sequence>
        <xsd:attribute name="expressionLanguage" type="xsd:anyURI" />
        <xsd:attribute name="logicalPeopleGroup" type="xsd:NCName" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tArgument">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleMixedContentElements">
        <xsd:attribute name="name" type="xsd:NCName" />
        <xsd:attribute name="expressionLanguage" type="xsd:anyURI" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tParameter" mixed="true">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleMixedContentElements">
        <xsd:attribute name="name" type="xsd:NCName" use="required" />
        <xsd:attribute name="type" type="xsd:QName" use="required" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tLiteral" mixed="true">
    <xsd:sequence>
      <xsd:any namespace="##any" processContents="lax"/>
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax" />
  </xsd:complexType>

  <xsd:complexType name="tQuery" mixed="true">
    <xsd:complexContent>
      <xsd:extension base="tExtensibleMixedContentElements">
        <xsd:attribute name="part" />
        <xsd:attribute name="queryLanguage" type="xsd:anyURI" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tExpression" mixed="true">
```

```
    <xsd:complexContent>
      <xsd:extension base="tExtensibleMixedContentElements">
        <xsd:attribute name="expressionLanguage" type="xsd:anyURI" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="priority" type="tPriority-expr" />
  <xsd:complexType name="tPriority-expr" mixed="true">
    <xsd:complexContent mixed="true">
      <xsd:extension base="tExpression" />
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tBoolean-expr" mixed="true">
    <xsd:complexContent mixed="true">
      <xsd:extension base="tExpression" />
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tDuration-expr" mixed="true">
    <xsd:complexContent mixed="true">
      <xsd:extension base="tExpression" />
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tDeadline-expr" mixed="true">
    <xsd:complexContent mixed="true">
      <xsd:extension base="tExpression" />
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:simpleType name="tBoolean">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="yes" />
      <xsd:enumeration value="no" />
    </xsd:restriction>
  </xsd:simpleType>

</xsd:schema>
```

C. WS-HumanTask Data Types Schema

```
 <?xml version="1.0" encoding="UTF-8"?>
<!--
  Copyright (c) OASIS Open 2009. All Rights Reserved.
-->
<xsd:schema
  targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
humantask/types/200803"
  xmlns="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/types/200803"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  blockDefault="#all">

  <xsd:annotation>
    <xsd:documentation>
      XML Schema for WS-HumanTask 1.1 - WS-HumanTask Data Type Definitions
    </xsd:documentation>
  </xsd:annotation>

  <!-- other namespaces -->
  <xsd:import namespace="http://www.w3.org/XML/1998/namespace"
schemaLocation="http://www.w3.org/2001/xml.xsd"/>

  <!-- data types for attachment operations -->
  <xsd:element name="attachmentInfo" type="tAttachmentInfo"/>
  <xsd:complexType name="tAttachmentInfo">
    <xsd:sequence>
      <xsd:element name="identifier" type="xsd:anyURI"/>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="accessType" type="xsd:string"/>
      <xsd:element name="contentType" type="xsd:string"/>
      <xsd:element name="contentCategory" type="xsd:anyURI"/>
      <xsd:element name="attachedTime" type="xsd:dateTime"/>
      <xsd:element name="attachedBy" type="tUser"/>
      <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="attachment" type="tAttachment"/>
  <xsd:complexType name="tAttachment">
    <xsd:sequence>
      <xsd:element ref="attachmentInfo"/>
      <xsd:element name="value" type="xsd:anyType"/>
    </xsd:sequence>
  </xsd:complexType>

  <!-- data types for comments -->
  <xsd:element name="comment" type="tComment"/>
  <xsd:complexType name="tComment">
    <xsd:sequence>
      <xsd:element name="addedTime" type="xsd:dateTime"/>
      <xsd:element name="addedBy" type="tUser"/>
      <xsd:element name="text" type="xsd:string"/>
      <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

  <!-- data types for simple query operations -->
  <xsd:element name="taskAbstract" type="tTaskAbstract"/>
  <xsd:complexType name="tTaskAbstract">
    <xsd:sequence>
      <xsd:element name="id" type="xsd:string"/>
      <xsd:element name="taskType" type="xsd:string"/>
      <xsd:element name="name" type="xsd:QName"/>
      <xsd:element name="status" type="tStatus"/>
      <xsd:element name="priority" type="tPriority" minOccurs="0"/>
      <xsd:element name="createdTime" type="xsd:dateTime"/>
      <xsd:element name="activationTime" type="xsd:dateTime" minOccurs="0"/>
      <xsd:element name="expirationTime" type="xsd:dateTime" minOccurs="0"/>
      <xsd:element name="isSkipable" type="xsd:boolean" minOccurs="0"/>
      <xsd:element name="hasPotentialOwners" type="xsd:boolean"
minOccurs="0"/>
      <xsd:element name="startByTimeExists" type="xsd:boolean"
minOccurs="0"/>
      <xsd:element name="completeByTimeExists" type="xsd:boolean"
minOccurs="0"/>
      <xsd:element name="presentationName" type="tPresentationName"
minOccurs="0"/>
      <xsd:element name="presentationSubject" type="tPresentationSubject"
minOccurs="0"/>
      <xsd:element name="renderingMethodExists" type="xsd:boolean"/>
      <xsd:element name="hasOutput" type="xsd:boolean" minOccurs="0"/>
      <xsd:element name="hasFault" type="xsd:boolean" minOccurs="0"/>
      <xsd:element name="hasAttachments" type="xsd:boolean" minOccurs="0"/>
      <xsd:element name="hasComments" type="xsd:boolean" minOccurs="0"/>
      <xsd:element name="escalated" type="xsd:boolean" minOccurs="0"/>
      <xsd:element name="outcome" type="xsd:string" minOccurs="0"/>
      <xsd:element name="parentTaskId" type="xsd:string" minOccurs="0"/>
      <xsd:element name="hasSubTasks" type="xsd:boolean" minOccurs="0"/>
      <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="taskDetails" type="tTaskDetails"/>
  <xsd:complexType name="tTaskDetails">
    <xsd:sequence>
      <xsd:element name="id" type="xsd:string"/>
      <xsd:element name="taskType" type="xsd:string"/>
      <xsd:element name="name" type="xsd:QName"/>
      <xsd:element name="status" type="tStatus"/>
      <xsd:element name="priority" type="tPriority" minOccurs="0"/>
      <xsd:element name="taskInitiator" type="tUser" minOccurs="0"/>
      <xsd:element name="taskStakeholders" type="tOrganizationalEntity"
minOccurs="0"/>
      <xsd:element name="potentialOwners" type="tOrganizationalEntity"
minOccurs="0"/>
      <xsd:element name="businessAdministrators" type="tOrganizationalEntity"
minOccurs="0"/>
      <xsd:element name="actualOwner" type="tUser" minOccurs="0"/>
      <xsd:element name="notificationRecipients" type="tOrganizationalEntity"
minOccurs="0"/>
      <xsd:element name="createdTime" type="xsd:dateTime"/>
      <xsd:element name="createdBy" type="xsd:string" minOccurs="0"/>
      <xsd:element name="lastModifiedTime" type="xsd:dateTime"/>
      <xsd:element name="lastModifiedBy" type="xsd:string" minOccurs="0"/>
      <xsd:element name="activationTime" type="xsd:dateTime" minOccurs="0"/>
      <xsd:element name="expirationTime" type="xsd:dateTime" minOccurs="0"/>
      <xsd:element name="isSkipable" type="xsd:boolean" minOccurs="0"/>
      <xsd:element name="hasPotentialOwners" type="xsd:boolean"
minOccurs="0"/>
      <xsd:element name="startByTimeExists" type="xsd:boolean"
minOccurs="0"/>
      <xsd:element name="completeByTimeExists" type="xsd:boolean"
minOccurs="0"/>
      <xsd:element name="presentationName" type="tPresentationName"
minOccurs="0"/>
      <xsd:element name="presentationSubject" type="tPresentationSubject"
minOccurs="0"/>
      <xsd:element name="renderingMethodExists" type="xsd:boolean"/>
      <xsd:element name="hasOutput" type="xsd:boolean" minOccurs="0"/>
      <xsd:element name="hasFault" type="xsd:boolean" minOccurs="0"/>
      <xsd:element name="hasAttachments" type="xsd:boolean" minOccurs="0"/>
      <xsd:element name="hasComments" type="xsd:boolean" minOccurs="0"/>
      <xsd:element name="escalated" type="xsd:boolean" minOccurs="0"/>
      <xsd:element name="searchBy" type="xsd:string" minOccurs="0"/>
      <xsd:element name="outcome" type="xsd:string" minOccurs="0"/>
      <xsd:element name="parentTaskId" type="xsd:string" minOccurs="0"/>
      <xsd:element name="hasSubTasks" type="xsd:boolean" minOccurs="0"/>
      <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:simpleType name="tPresentationName">
    <xsd:annotation>
      <xsd:documentation>length-restricted string</xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="64"/>
      <xsd:whiteSpace value="preserve"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="tPresentationSubject">
    <xsd:annotation>
      <xsd:documentation>length-restricted string</xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="254"/>
      <xsd:whiteSpace value="preserve"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="tStatus">
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>
  <xsd:simpleType name="tPredefinedStatus">
    <xsd:annotation>
      <xsd:documentation>for documentation only</xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="CREATED"/>
      <xsd:enumeration value="READY"/>
      <xsd:enumeration value="RESERVED"/>
      <xsd:enumeration value="IN_PROGRESS"/>
      <xsd:enumeration value="SUSPENDED"/>
      <xsd:enumeration value="COMPLETED"/>
      <xsd:enumeration value="FAILED"/>
      <xsd:enumeration value="ERROR"/>
      <xsd:enumeration value="EXITED"/>
      <xsd:enumeration value="OBSOLETE"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="tPriority">
    <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="0"/>
      <xsd:maxInclusive value="10"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:complexType name="tTime">
    <xsd:choice>
      <xsd:element name="timePeriod" type="xsd:duration"/>
      <xsd:element name="pointOfTime" type="xsd:dateTime"/>
    </xsd:choice>
  </xsd:complexType>

  <!-- task operations -->
  <xsd:complexType name="tTaskOperations">
    <xsd:choice maxOccurs="unbounded">
      <xsd:element name="claim" type="tTaskOperation"/>
      <xsd:element name="start" type="tTaskOperation"/>
      <xsd:element name="stop" type="tTaskOperation"/>
      <xsd:element name="release" type="tTaskOperation"/>
      <xsd:element name="suspend" type="tTaskOperation"/>
      <xsd:element name="suspendUntil" type="tTaskOperation"/>
      <xsd:element name="resume" type="tTaskOperation"/>
      <xsd:element name="complete" type="tTaskOperation"/>
      <xsd:element name="remove" type="tTaskOperation"/>
      <xsd:element name="fail" type="tTaskOperation"/>
      <xsd:element name="setPriority" type="tTaskOperation"/>
      <xsd:element name="addAttachment" type="tTaskOperation"/>
      <xsd:element name="getAttachmentInfos" type="tTaskOperation"/>
      <xsd:element name="getAttachment" type="tTaskOperation"/>
      <xsd:element name="deleteAttachment" type="tTaskOperation"/>
      <xsd:element name="addComment" type="tTaskOperation"/>
      <xsd:element name="updateComment" type="tTaskOperation"/>
      <xsd:element name="deleteComment" type="tTaskOperation"/>
      <xsd:element name="getComments" type="tTaskOperation"/>
      <xsd:element name="skip" type="tTaskOperation"/>
      <xsd:element name="forward" type="tTaskOperation"/>
      <xsd:element name="delegate" type="tTaskOperation"/>
      <xsd:element name="getRendering" type="tTaskOperation"/>
      <xsd:element name="getRenderingTypes" type="tTaskOperation"/>
      <xsd:element name="getTaskDetails" type="tTaskOperation"/>
      <xsd:element name="getTaskDescription" type="tTaskOperation"/>
      <xsd:element name="setOutput" type="tTaskOperation"/>
      <xsd:element name="deleteOutput" type="tTaskOperation"/>
      <xsd:element name="setFault" type="tTaskOperation"/>
      <xsd:element name="deleteFault" type="tTaskOperation"/>
      <xsd:element name="getInput" type="tTaskOperation"/>
      <xsd:element name="getOutput" type="tTaskOperation"/>
      <xsd:element name="getFault" type="tTaskOperation"/>
      <xsd:element name="getOutcome" type="tTaskOperation"/>
      <xsd:element name="getTaskHistory" type="tTaskOperation"/>
      <xsd:element name="getTaskInstanceData" type="tTaskOperation"/>
      <xsd:element name="getSubtasks" type="tTaskOperation"/>
      <xsd:element name="getSubtaskIdentifiers" type="tTaskOperation"/>
      <xsd:element name="hasSubtasks" type="tTaskOperation"/>
      <xsd:element name="getParentTask" type="tTaskOperation"/>
      <xsd:element name="getParentTaskIdentifier" type="tTaskOperation"/>
      <xsd:element name="isSubtask" type="tTaskOperation"/>
      <xsd:element name="instantiateSubtask" type="tTaskOperation"/>
      <xsd:element name="setTaskStartDeadlineExpression"
type="tTaskOperation"/>
      <xsd:element name="setTaskStartDurationExpression"
type="tTaskOperation"/>
      <xsd:element name="setTaskCompletionDeadlineExpression"
type="tTaskOperation"/>
      <xsd:element name="setTaskCompletionDurationExpression"
type="tTaskOperation"/>
      <xsd:element name="activate" type="tTaskOperation"/>
      <xsd:element name="nominate" type="tTaskOperation"/>
      <xsd:element name="setGenericHumanRole" type="tTaskOperation"/>
      <xsd:any namespace="##other" processContents="lax"/>
    </xsd:choice>
  </xsd:complexType>
  <xsd:complexType name="tTaskOperation">
    <xsd:complexContent>
      <xsd:restriction base="xsd:anyType"/>
    </xsd:complexContent>
  </xsd:complexType>

  <!-- data types for advanced query operations -->
  <xsd:element name="taskQueryResultSet" type="tTaskQueryResultSet"/>
  <xsd:complexType name="tTaskQueryResultSet">
    <xsd:sequence>
      <xsd:element name="row" type="tTaskQueryResultRow" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="tTaskQueryResultRow">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="id" type="xsd:string"/>
      <xsd:element name="taskType" type="xsd:string"/>
      <xsd:element name="name" type="xsd:QName"/>
      <xsd:element name="status" type="tStatus"/>
      <xsd:element name="priority" type="tPriority"/>
      <xsd:element name="taskInitiator" type="tOrganizationalEntity"/>
      <xsd:element name="taskStakeholders" type="tOrganizationalEntity"/>
      <xsd:element name="potentialOwners" type="tOrganizationalEntity"/>
      <xsd:element name="businessAdministrators"
type="tOrganizationalEntity"/>
      <xsd:element name="actualOwner" type="tUser"/>
      <xsd:element name="notificationRecipients"
type="tOrganizationalEntity"/>
      <xsd:element name="createdTime" type="xsd:dateTime"/>
      <xsd:element name="createdBy" type="xsd:string"/>
      <xsd:element name="lastModifiedTime" type="xsd:dateTime"/>
      <xsd:element name="lastModifiedBy" type="xsd:string"/>
      <xsd:element name="activationTime" type="xsd:dateTime"/>
      <xsd:element name="expirationTime" type="xsd:dateTime"/>
      <xsd:element name="isSkipable" type="xsd:boolean"/>
      <xsd:element name="hasPotentialOwners" type="xsd:boolean"/>
      <xsd:element name="startByTime" type="xsd:dateTime"/>
      <xsd:element name="completeByTime" type="xsd:dateTime"/>
      <xsd:element name="presentationName" type="tPresentationName"/>
      <xsd:element name="presentationSubject" type="tPresentationSubject"/>
      <xsd:element name="renderingMethodName" type="xsd:QName"/>
      <xsd:element name="hasOutput" type="xsd:boolean"/>
      <xsd:element name="hasFault" type="xsd:boolean"/>
      <xsd:element name="hasAttachments" type="xsd:boolean"/>
      <xsd:element name="hasComments" type="xsd:boolean"/>
      <xsd:element name="escalated" type="xsd:boolean"/>
      <xsd:element name="parentTaskId" type="xsd:string"/>
      <xsd:element name="hasSubtasks" type="xsd:boolean"/>
      <xsd:element name="searchBy" type="xsd:string"/>
      <xsd:element name="outcome" type="xsd:string"/>
      <xsd:element name="taskOperations" type="tTaskOperations"/>
      <xsd:any namespace="##other" processContents="lax"/>
    </xsd:choice>
  </xsd:complexType>
  <xsd:complexType name="tFault">
    <xsd:sequence>
      <xsd:element name="faultName" type="xsd:NCName"/>
      <xsd:element name="faultData" type="xsd:anyType"/>
    </xsd:sequence>
  </xsd:complexType>

  <!-- elements and types for organizational entities -->
  <xsd:element name="organizationalEntity" type="tOrganizationalEntity"/>
  <xsd:complexType name="tOrganizationalEntity">
    <xsd:choice maxOccurs="unbounded">
      <xsd:element name="user" type="tUser"/>
      <xsd:element name="group" type="tGroup"/>
    </xsd:choice>
  </xsd:complexType>
  <xsd:element name="user" type="tUser"/>
  <xsd:simpleType name="tUser">
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>
  <xsd:element name="group" type="tGroup"/>
  <xsd:simpleType name="tGroup">
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>

  <!--  input or output message part data  -->
  <xsd:element name="part" type="tPart"/>
  <xsd:complexType name="tPart" mixed="true">
    <xsd:sequence>
      <xsd:any processContents="skip" minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:NCName" use="required"/>
  </xsd:complexType>

  <!--  type container element for one or more message parts -->
  <xsd:complexType name="tMessagePartsData">
    <xsd:sequence>
      <xsd:element ref="part" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="tFaultData">
    <xsd:sequence>
      <xsd:element name="faultName" type="xsd:NCName"/>
      <xsd:element name="faultData" type="xsd:anyType"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="attachmentInfos" type="tAttachmentInfos"/>
  <xsd:complexType name="tAttachmentInfos">
    <xsd:sequence>
      <xsd:element name="info" type="tAttachmentInfo" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="comments" type="tComments"/>
  <xsd:complexType name="tComments">
    <xsd:sequence>
      <xsd:element ref="comment" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="renderingType" type="xsd:QName"/>
  <xsd:complexType name="tRenderingTypes">
    <xsd:sequence>
      <xsd:element ref="renderingType" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

  <!--  Single rendering element that contains rendering type (attribute) and
data. -->
  <xsd:element name="rendering" type="tRendering"/>
  <xsd:complexType name="tRendering">
    <xsd:sequence>
      <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="type" type="xsd:QName" use="required"/>
  </xsd:complexType>
  <xsd:element name="renderings">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="rendering" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="description" type="xsd:string"/>
  <xsd:complexType name="tTaskInstanceData">
    <xsd:sequence>
      <!--  taskDetails contains task ID, meta data, presentation name and
presentation subject. -->
      <xsd:element ref="taskDetails"/>
      <xsd:element ref="description"/>
      <xsd:element name="input" type="tMessagePartsData"/>
      <xsd:element name="output" type="tMessagePartsData" nillable="true"/>
      <xsd:element name="fault" type="tFaultData" nillable="true"
minOccurs="0"/>
      <xsd:element ref="renderings" minOccurs="0"/>
      <xsd:element ref="comments" minOccurs="0"/>
      <xsd:element ref="attachmentInfos" minOccurs="0"/>
      <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

  <!--  Defines the human task event types -->
  <xsd:simpleType name="tTaskEventType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="create"/>
      <xsd:enumeration value="claim"/>
      <xsd:enumeration value="start"/>
      <xsd:enumeration value="stop"/>
      <xsd:enumeration value="release"/>
      <xsd:enumeration value="suspend"/>
      <xsd:enumeration value="suspendUntil"/>
      <xsd:enumeration value="resume"/>
      <xsd:enumeration value="complete"/>
      <xsd:enumeration value="remove"/>
      <xsd:enumeration value="fail"/>
      <xsd:enumeration value="setPriority"/>
      <xsd:enumeration value="addAttachment"/>
      <xsd:enumeration value="deleteattachment"/>
      <xsd:enumeration value="addComment"/>
      <xsd:enumeration value="skip"/>
      <xsd:enumeration value="forward"/>
      <xsd:enumeration value="delegate"/>
      <xsd:enumeration value="setOutput"/>
      <xsd:enumeration value="deleteOutput"/>
      <xsd:enumeration value="setFault"/>
      <xsd:enumeration value="deleteFault"/>
      <xsd:enumeration value="activate"/>
      <xsd:enumeration value="nominate"/>
      <xsd:enumeration value="setGenericHumanRole"/>
      <xsd:enumeration value="expire"/>
      <xsd:enumeration value="escalated"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:element name="taskEvent">
    <xsd:complexType>
      <xsd:annotation>
        <xsd:documentation>
               A detailed event that represnts a change in the task's state.
            </xsd:documentation>
      </xsd:annotation>
      <xsd:sequence>
        <!--  event id - unique per task -->
        <xsd:element name="id" type="xsd:integer"/>
        <!--  event  date time -->
        <xsd:element name="eventTime" type="xsd:dateTime"/>
        <!--  task ID -->
        <xsd:element name="identifier" type="xsd:anyURI"/>
        <xsd:element name="principal" type="xsd:string" nillable="true"
minOccurs="0"/>
```

```
        <!--  Event type. Note - using a restricted type limits extensibility
to add custom event types. -->
        <xsd:element name="eventType" type="tTaskEventType"/>
        <!--  actual owner of the task before the event -->
        <xsd:element name="startOwner" type="xsd:string" nillable="true"
minOccurs="0"/>
        <!--  actual owner of the task after the event -->
        <xsd:element name="endOwner" type="xsd:string" nillable="true"
minOccurs="0"/>
        <!--  WSHT task status -->
        <xsd:element name="status" type="tStatus"/>
        <!-- boolean to indicate this event has optional data -->
        <xsd:element name="hasData" type="xsd:boolean" minOccurs="0"/>
        <xsd:element name="eventData" type="xsd:anyType" nillable="true"
minOccurs="0"/>
        <xsd:element name="faultName" type="xsd:string" nillable="true"
minOccurs="0"/>
        <!-- extensibility -->
        <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <!--  Filter allow list event by eventId or other params such as status and
event type -->
  <xsd:complexType name="tTaskHistoryFilter">
    <xsd:choice>
      <xsd:element name="eventId" type="xsd:integer"/>
      <!--  Filter to allow narrow down query by status, principal, event
Type. -->
      <xsd:sequence>
        <xsd:element name="status" type="tStatus" minOccurs="0"
maxOccurs="unbounded"/>
        <xsd:element name="eventType" type="tTaskEventType" minOccurs="0"
maxOccurs="unbounded"/>
        <xsd:element name="principal" type="xsd:string" minOccurs="0"/>
        <xsd:element name="afterEventTime" type="xsd:dateTime"
minOccurs="0"/>
        <xsd:element name="beforeEventTime" type="xsd:dateTime"
minOccurs="0"/>
      </xsd:sequence>
    </xsd:choice>
  </xsd:complexType>
</xsd:schema>
```

D. WS-HumanTask Client API Port Type

```
 <?xml version="1.0" encoding="UTF-8"?>
<!--
  Copyright (c) OASIS Open 2009. All Rights Reserved.
-->
<wsdl:definitions
  targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
humantask/api/200803"
  xmlns="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/api/200803"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:htt="http://docs.oasis-open.org/ns/bpel4people/ws-
humantask/types/200803">

  <wsdl:documentation>
    Web Service Definition for WS-HumanTask 1.1 - Operations for Client
Applications
  </wsdl:documentation>

  <wsdl:types>
    <xsd:schema
      targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
humantask/api/200803"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:htt="http://docs.oasis-open.org/ns/bpel4people/ws-
humantask/types/200803"
      elementFormDefault="qualified"
      blockDefault="#all">

      <xsd:import
        namespace="http://docs.oasis-open.org/ns/bpel4people/ws-
humantask/types/200803"
        schemaLocation="ws-humantask-types.xsd"/>

      <!-- Input and output elements -->
      <xsd:element name="claim">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="claimResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:annotation>
              <xsd:documentation>Empty message</xsd:documentation>
            </xsd:annotation>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="batchClaim">
        <xsd:complexType>
```

```
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"
maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="batchClaimResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="batchResponse" type="tBatchResponse"
minOccurs="0" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="start">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="startResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:annotation>
              <xsd:documentation>Empty message</xsd:documentation>
            </xsd:annotation>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="batchStart">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"
maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="batchStartResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="batchResponse" type="tBatchResponse"
minOccurs="0" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="stop">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="stopResponse">
        <xsd:complexType>
```

```
          <xsd:sequence>
            <xsd:annotation>
              <xsd:documentation>Empty message</xsd:documentation>
            </xsd:annotation>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="batchStop">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"
maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="batchStopResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="batchResponse" type="tBatchResponse"
minOccurs="0" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="release">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="releaseResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:annotation>
              <xsd:documentation>Empty message</xsd:documentation>
            </xsd:annotation>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="batchRelease">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"
maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="batchReleaseResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="batchResponse" type="tBatchResponse"
minOccurs="0" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="suspend">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="suspendResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:annotation>
              <xsd:documentation>Empty message</xsd:documentation>
            </xsd:annotation>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="batchSuspend">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"
maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="batchSuspendResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="batchResponse" type="tBatchResponse"
minOccurs="0" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="suspendUntil">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
            <xsd:element name="time" type="htt:tTime"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="suspendUntilResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:annotation>
              <xsd:documentation>Empty message</xsd:documentation>
            </xsd:annotation>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="batchSuspendUntil">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"
maxOccurs="unbounded"/>
```

```
            <xsd:element name="time" type="htt:tTime"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="batchSuspendUntilResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="batchResponse" type="tBatchResponse"
minOccurs="0" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="resume">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="resumeResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:annotation>
              <xsd:documentation>Empty message</xsd:documentation>
            </xsd:annotation>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="batchResume">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"
maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="batchResumeResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="batchResponse" type="tBatchResponse"
minOccurs="0" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="complete">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
            <xsd:element name="taskData" type="xsd:anyType" minOccurs="0"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="completeResponse">
        <xsd:complexType>
          <xsd:sequence>
```

```
            <xsd:annotation>
              <xsd:documentation>Empty message</xsd:documentation>
            </xsd:annotation>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="batchComplete">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"
maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="batchCompleteResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="batchResponse" type="tBatchResponse"
minOccurs="0" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="remove">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="removeResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:annotation>
              <xsd:documentation>Empty message</xsd:documentation>
            </xsd:annotation>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="batchRemove">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"
maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="batchRemoveResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="batchResponse" type="tBatchResponse"
minOccurs="0" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="fail">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
            <xsd:element name="fault" type="htt:tFault" minOccurs="0"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="failResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:annotation>
              <xsd:documentation>Empty message</xsd:documentation>
            </xsd:annotation>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="batchFail">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"
maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="batchFailResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="batchResponse" type="tBatchResponse"
minOccurs="0" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="setPriority">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
            <xsd:element name="priority" type="htt:tPriority"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="setPriorityResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:annotation>
              <xsd:documentation>Empty message</xsd:documentation>
            </xsd:annotation>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="batchSetPriority">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"
maxOccurs="unbounded"/>
```

```
            <xsd:element name="priority" type="htt:tPriority"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="batchSetPriorityResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="batchResponse" type="tBatchResponse"
minOccurs="0" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="addAttachment">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="taskIdentifier" type="xsd:anyURI"/>
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element name="accessType" type="xsd:string"/>
            <xsd:element name="contentType" type="xsd:string"/>
            <xsd:element name="attachment" type="xsd:anyType"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="addAttachmentResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="getAttachmentInfos">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="getAttachmentInfosResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="info" type="htt:tAttachmentInfo" minOccurs="0"
maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="getAttachment">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="taskIdentifier" type="xsd:anyURI"/>
            <xsd:element name="attachmentIdentifier" type="xsd:anyURI"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="getAttachmentResponse">
        <xsd:complexType>
```

```
          <xsd:sequence>
            <xsd:element name="attachment" type="htt:tAttachment"
minOccurs="0" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="deleteAttachment">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="taskIdentifier" type="xsd:anyURI"/>
            <xsd:element name="attachmentIdentifier" type="xsd:anyURI"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="deleteAttachmentResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:annotation>
              <xsd:documentation>Empty message</xsd:documentation>
            </xsd:annotation>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="addComment">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
            <xsd:element name="text" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="addCommentResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="commentID" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="updateComment">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="taskIdentifier" type="xsd:anyURI"/>
            <xsd:element name="commentIdentifier" type="xsd:anyURI"/>
            <xsd:element name="text" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="updateCommentResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:annotation>
              <xsd:documentation>Empty message</xsd:documentation>
            </xsd:annotation>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="deleteComment">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="taskIdentifier" type="xsd:anyURI"/>
            <xsd:element name="commentIdentifier" type="xsd:anyURI"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="deleteCommentResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:annotation>
              <xsd:documentation>Empty message</xsd:documentation>
            </xsd:annotation>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="getComments">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="getCommentsResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="comment" type="htt:tComment" minOccurs="0"
maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="skip">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="skipResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:annotation>
              <xsd:documentation>Empty message</xsd:documentation>
            </xsd:annotation>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="batchSkip">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"
maxOccurs="unbounded"/>
```

```
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="batchSkipResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="batchResponse" type="tBatchResponse"
minOccurs="0" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="forward">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
            <xsd:element name="organizationalEntity"
type="htt:tOrganizationalEntity"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="forwardResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:annotation>
              <xsd:documentation>Empty message</xsd:documentation>
            </xsd:annotation>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="batchForward">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"
maxOccurs="unbounded"/>
            <xsd:element name="organizationalEntity"
type="htt:tOrganizationalEntity"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="batchForwardResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="batchResponse" type="tBatchResponse"
minOccurs="0" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="delegate">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
            <xsd:element name="organizationalEntity"
type="htt:tOrganizationalEntity"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="delegateResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:annotation>
              <xsd:documentation>Empty message</xsd:documentation>
            </xsd:annotation>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="batchDelegate">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"
maxOccurs="unbounded"/>
            <xsd:element name="organizationalEntity"
type="htt:tOrganizationalEntity"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="batchDelegateResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="batchResponse" type="tBatchResponse"
minOccurs="0" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="getRendering">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyType"/>
            <xsd:element name="renderingType" type="xsd:QName"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="getRenderingResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="rendering" type="xsd:anyType"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="getRenderingTypes">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyType"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="getRenderingTypesResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="renderingType" type="xsd:QName" minOccurs="0"
maxOccurs="unbounded"/>
```

```
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="getTaskDetails">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="getTaskDetailsResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="taskDetails" type="htt:tTaskDetails"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="getTaskDescription">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
            <xsd:element name="contentType" type="xsd:string" minOccurs="0"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="getTaskDescriptionResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="description" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="setOutput">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
            <xsd:element name="part" type="xsd:NCName" minOccurs="0"/>
            <xsd:element name="taskData" type="xsd:anyType"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="setOutputResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:annotation>
              <xsd:documentation>Empty message</xsd:documentation>
            </xsd:annotation>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="deleteOutput">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="deleteOutputResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:annotation>
              <xsd:documentation>Empty message</xsd:documentation>
            </xsd:annotation>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="setFault">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
            <xsd:element name="fault" type="htt:tFault"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="setFaultResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:annotation>
              <xsd:documentation>Empty message</xsd:documentation>
            </xsd:annotation>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="deleteFault">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="deleteFaultResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:annotation>
              <xsd:documentation>Empty message</xsd:documentation>
            </xsd:annotation>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="getInput">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
            <xsd:element name="part" type="xsd:NCName" minOccurs="0"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="getInputResponse">
        <xsd:complexType>
```

```
          <xsd:sequence>
            <xsd:element name="taskData" type="xsd:anyType"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="getOutput">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
            <xsd:element name="part" type="xsd:NCName" minOccurs="0"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="getOutputResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="taskData" type="xsd:anyType"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="getFault">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="getFaultResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="fault" type="htt:tFault"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="getMyTaskAbstracts">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="taskType" type="xsd:string"/>
            <xsd:element name="genericHumanRole" type="xsd:string"
minOccurs="0"/>
            <xsd:element name="workQueue" type="xsd:string" minOccurs="0"/>
            <xsd:element name="status" type="htt:tStatus" minOccurs="0"
maxOccurs="unbounded"/>
            <xsd:element name="whereClause" type="xsd:string" minOccurs="0"/>
            <xsd:element name="createdOnClause" type="xsd:string"
minOccurs="0"/>
            <xsd:element name="maxTasks" type="xsd:int" minOccurs="0"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="getMyTaskAbstractsResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="taskAbstract" type="htt:tTaskAbstract"
minOccurs="0" maxOccurs="unbounded"/>
```

```
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="getMyTaskDetails">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="taskType" type="xsd:string"/>
            <xsd:element name="genericHumanRole" type="xsd:string"
minOccurs="0"/>
            <xsd:element name="workQueue" type="xsd:string" minOccurs="0"/>
            <xsd:element name="status" type="htt:tStatus" minOccurs="0"
maxOccurs="unbounded"/>
            <xsd:element name="whereClause" type="xsd:string" minOccurs="0"/>
            <xsd:element name="createdOnClause" type="xsd:string"
minOccurs="0"/>
            <xsd:element name="maxTasks" type="xsd:int" minOccurs="0"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="getMyTaskDetailsResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="taskDetails" type="htt:tTaskDetails"
minOccurs="0" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="query">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="selectClause" type="xsd:string"/>
            <xsd:element name="whereClause" type="xsd:string" minOccurs="0"/>
            <xsd:element name="orderByClause" type="xsd:string"
minOccurs="0"/>
            <xsd:element name="maxTasks" type="xsd:int" minOccurs="0"/>
            <xsd:element name="taskIndexOffset" type="xsd:int"
minOccurs="0"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="queryResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="taskQueryResultSet"
type="htt:tTaskQueryResultSet"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="activate">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="activateResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:annotation>
              <xsd:documentation>Empty message</xsd:documentation>
            </xsd:annotation>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="batchActivate">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"
maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="batchActivateResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="batchResponse" type="tBatchResponse"
minOccurs="0" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="nominate">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
            <xsd:element name="organizationalEntity"
type="htt:tOrganizationalEntity"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="nominateResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:annotation>
              <xsd:documentation>Empty message</xsd:documentation>
            </xsd:annotation>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="batchNominate">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"
maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="batchNominateResponse">
        <xsd:complexType>
          <xsd:sequence>
```

```
            <xsd:element name="batchResponse" type="tBatchResponse"
minOccurs="0" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="setGenericHumanRole">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
            <xsd:element name="genericHumanRole" type="xsd:string"/>
            <xsd:element name="organizationalEntity"
type="htt:tOrganizationalEntity"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="setGenericHumanRoleResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:annotation>
              <xsd:documentation>Empty message</xsd:documentation>
            </xsd:annotation>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="batchSetGenericHumanRole">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"
maxOccurs="unbounded"/>
            <xsd:element name="genericHumanRole" type="xsd:string"/>
            <xsd:element name="organizationalEntity"
type="htt:tOrganizationalEntity"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="batchSetGenericHumanRoleResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="batchResponse" type="tBatchResponse"
minOccurs="0" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>


      <xsd:element name="getOutcome">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="getOutcomeResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="outcome" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="getTaskOperations">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="getTaskOperationsResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="taskOperations" type="htt:tTaskOperations"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="getTaskInstanceData">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
            <xsd:element name="properties" type="xsd:string"/>
            <xsd:element name="renderingPreferences"
type="htt:tRenderingTypes" minOccurs="0" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="getTaskInstanceDataResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="taskInstanceData"
type="htt:tTaskInstanceData"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="getTaskHistory">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
            <xsd:element name="filter" type="htt:tTaskHistoryFilter"
minOccurs="0"/>
            <xsd:element name="startIndex" type="xsd:integer" minOccurs="0"/>
            <xsd:element name="maxTasks" type="xsd:integer" minOccurs="0"/>
          </xsd:sequence>
          <xsd:attribute name="includeData" type="xsd:boolean"/>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="getTaskHistoryResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="taskEvent" type="htt:tTaskEventType"
minOccurs="0" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="setTaskStartDeadlineExpression">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
            <xsd:element name="deadlineName" type="xsd:NCName"/>
            <xsd:element name="deadlineExpression" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="setTaskStartDeadlineExpressionResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:annotation>
              <xsd:documentation>Empty message</xsd:documentation>
            </xsd:annotation>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="setTaskStartDurationExpression">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
            <xsd:element name="deadlineName" type="xsd:NCName"/>
            <xsd:element name="durationExpression" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="setTaskStartDurationExpressionResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:annotation>
              <xsd:documentation>Empty message</xsd:documentation>
            </xsd:annotation>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="setTaskCompletionDeadlineExpression">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
            <xsd:element name="deadlineName" type="xsd:NCName"/>
            <xsd:element name="deadlineExpression" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="setTaskCompletionDeadlineExpressionResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:annotation>
              <xsd:documentation>Empty message</xsd:documentation>
            </xsd:annotation>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="setTaskCompletionDurationExpression">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="identifier" type="xsd:anyURI"/>
            <xsd:element name="deadlineName" type="xsd:NCName"/>
            <xsd:element name="durationExpression" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="setTaskCompletionDurationExpressionResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:annotation>
              <xsd:documentation>Empty message</xsd:documentation>
            </xsd:annotation>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <!-- Fault elements -->
      <xsd:element name="illegalState">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="status" type="htt:tStatus"/>
            <xsd:element name="message" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="illegalArgument" type="xsd:string"/>

      <xsd:element name="illegalAccess" type="xsd:string"/>

      <xsd:element name="illegalOperation" type="xsd:string"/>

      <xsd:element name="recipientNotAllowed" type="xsd:string"/>

      <xsd:complexType name="tBatchResponse">
        <xsd:sequence>
          <xsd:element name="identifier" type="xsd:anyURI"/>
          <xsd:choice>
            <xsd:element ref="illegalState"/>
            <xsd:element ref="illegalArgument"/>
            <xsd:element ref="illegalAccess"/>
            <xsd:element ref="illegalOperation"/>
            <xsd:element ref="recipientNotAllowed"/>
            <xsd:any namespace="##other" processContents="lax"/>
          </xsd:choice>
        </xsd:sequence>
      </xsd:complexType>

    </xsd:schema>
  </wsdl:types>

  <!-- Declaration of messages -->
  <wsdl:message name="claim">
    <wsdl:part name="claim" element="claim"/>
  </wsdl:message>
  <wsdl:message name="claimResponse">
    <wsdl:part name="claimResponse" element="claimResponse"/>
  </wsdl:message>

  <wsdl:message name="batchClaim">
    <wsdl:part name="batchClaim" element="batchClaim"/>
  </wsdl:message>
  <wsdl:message name="batchClaimResponse">
    <wsdl:part name="batchClaimResponse" element="batchClaimResponse"/>
  </wsdl:message>

  <wsdl:message name="start">
    <wsdl:part name="start" element="start"/>
  </wsdl:message>
  <wsdl:message name="startResponse">
    <wsdl:part name="startResponse" element="startResponse"/>
  </wsdl:message>

  <wsdl:message name="batchStart">
    <wsdl:part name="batchStart" element="batchStart"/>
  </wsdl:message>
  <wsdl:message name="batchStartResponse">
    <wsdl:part name="batchStartResponse" element="batchStartResponse"/>
  </wsdl:message>

  <wsdl:message name="stop">
    <wsdl:part name="stop" element="stop"/>
  </wsdl:message>
  <wsdl:message name="stopResponse">
    <wsdl:part name="stopResponse" element="stopResponse"/>
  </wsdl:message>

  <wsdl:message name="batchStop">
    <wsdl:part name="batchStop" element="batchStop"/>
  </wsdl:message>
  <wsdl:message name="batchStopResponse">
    <wsdl:part name="batchStopResponse" element="batchStopResponse"/>
  </wsdl:message>

  <wsdl:message name="release">
    <wsdl:part name="release" element="release"/>
  </wsdl:message>
  <wsdl:message name="releaseResponse">
    <wsdl:part name="releaseResponse" element="releaseResponse"/>
  </wsdl:message>

  <wsdl:message name="batchRelease">
    <wsdl:part name="batchRelease" element="batchRelease"/>
  </wsdl:message>
  <wsdl:message name="batchReleaseResponse">
    <wsdl:part name="batchReleaseResponse" element="batchReleaseResponse"/>
  </wsdl:message>

  <wsdl:message name="suspend">
    <wsdl:part name="suspend" element="suspend"/>
  </wsdl:message>
  <wsdl:message name="suspendResponse">
    <wsdl:part name="suspendResponse" element="suspendResponse"/>
  </wsdl:message>

  <wsdl:message name="batchSuspend">
    <wsdl:part name="batchSuspend" element="batchSuspend"/>
  </wsdl:message>
  <wsdl:message name="batchSuspendResponse">
    <wsdl:part name="batchSuspendResponse" element="batchSuspendResponse"/>
  </wsdl:message>

  <wsdl:message name="suspendUntil">
    <wsdl:part name="suspendUntil" element="suspendUntil"/>
  </wsdl:message>
  <wsdl:message name="suspendUntilResponse">
    <wsdl:part name="suspendUntilResponse" element="suspendUntilResponse"/>
  </wsdl:message>

  <wsdl:message name="batchSuspendUntil">
    <wsdl:part name="batchSuspendUntil" element="batchSuspendUntil"/>
  </wsdl:message>
  <wsdl:message name="batchSuspendUntilResponse">
    <wsdl:part name="batchSuspendUntilResponse"
element="batchSuspendUntilResponse"/>
  </wsdl:message>

  <wsdl:message name="resume">
    <wsdl:part name="resume" element="resume"/>
  </wsdl:message>
  <wsdl:message name="resumeResponse">
    <wsdl:part name="resumeResponse" element="resumeResponse"/>
  </wsdl:message>

  <wsdl:message name="batchResume">
    <wsdl:part name="batchResume" element="batchResume"/>
  </wsdl:message>
  <wsdl:message name="batchResumeResponse">
    <wsdl:part name="batchResumeResponse" element="batchResumeResponse"/>
  </wsdl:message>

  <wsdl:message name="complete">
    <wsdl:part name="complete" element="complete"/>
  </wsdl:message>
  <wsdl:message name="completeResponse">
    <wsdl:part name="completeResponse" element="completeResponse"/>
  </wsdl:message>

  <wsdl:message name="batchComplete">
    <wsdl:part name="batchComplete" element="batchComplete"/>
  </wsdl:message>
  <wsdl:message name="batchCompleteResponse">
    <wsdl:part name="batchCompleteResponse" element="batchCompleteResponse"/>
  </wsdl:message>

  <wsdl:message name="remove">
    <wsdl:part name="remove" element="remove"/>
  </wsdl:message>
  <wsdl:message name="removeResponse">
    <wsdl:part name="removeResponse" element="removeResponse"/>
  </wsdl:message>

  <wsdl:message name="batchRemove">
    <wsdl:part name="batchRemove" element="batchRemove"/>
  </wsdl:message>
  <wsdl:message name="batchRemoveResponse">
    <wsdl:part name="batchRemoveResponse" element="batchRemoveResponse"/>
  </wsdl:message>

  <wsdl:message name="fail">
    <wsdl:part name="fail" element="fail"/>
  </wsdl:message>
  <wsdl:message name="failResponse">
    <wsdl:part name="failResponse" element="failResponse"/>
  </wsdl:message>

  <wsdl:message name="batchFail">
    <wsdl:part name="batchFail" element="batchFail"/>
  </wsdl:message>
  <wsdl:message name="batchFailResponse">
    <wsdl:part name="batchFailResponse" element="batchFailResponse"/>
  </wsdl:message>

  <wsdl:message name="setPriority">
    <wsdl:part name="setPriority" element="setPriority"/>
  </wsdl:message>
  <wsdl:message name="setPriorityResponse">
    <wsdl:part name="setPriorityResponse" element="setPriorityResponse"/>
  </wsdl:message>

  <wsdl:message name="batchSetPriority">
    <wsdl:part name="batchSetPriority" element="batchSetPriority"/>
  </wsdl:message>
  <wsdl:message name="batchSetPriorityResponse">
    <wsdl:part name="batchSetPriorityResponse"
element="batchSetPriorityResponse"/>
  </wsdl:message>

  <wsdl:message name="addAttachment">
    <wsdl:part name="addAttachment" element="addAttachment"/>
  </wsdl:message>
  <wsdl:message name="addAttachmentResponse">
    <wsdl:part name="addAttachmentResponse" element="addAttachmentResponse"/>
  </wsdl:message>

  <wsdl:message name="getAttachmentInfos">
    <wsdl:part name="getAttachmentInfos" element="getAttachmentInfos"/>
  </wsdl:message>
  <wsdl:message name="getAttachmentInfosResponse">
    <wsdl:part name="getAttachmentInfosResponse"
element="getAttachmentInfosResponse"/>
  </wsdl:message>

  <wsdl:message name="getAttachment">
    <wsdl:part name="getAttachment" element="getAttachment"/>
  </wsdl:message>
  <wsdl:message name="getAttachmentResponse">
    <wsdl:part name="getAttachmentResponse" element="getAttachmentResponse"/>
  </wsdl:message>

  <wsdl:message name="deleteAttachment">
    <wsdl:part name="deleteAttachment" element="deleteAttachment"/>
  </wsdl:message>
  <wsdl:message name="deleteAttachmentResponse">
    <wsdl:part name="deleteAttachmentResponse"
element="deleteAttachmentResponse"/>
  </wsdl:message>

  <wsdl:message name="addComment">
    <wsdl:part name="addComment" element="addComment"/>
  </wsdl:message>
  <wsdl:message name="addCommentResponse">
    <wsdl:part name="addCommentResponse" element="addCommentResponse"/>
  </wsdl:message>

  <wsdl:message name="getComments">
    <wsdl:part name="getComments" element="getComments"/>
  </wsdl:message>
  <wsdl:message name="getCommentsResponse">
    <wsdl:part name="getCommentsResponse" element="getCommentsResponse"/>
  </wsdl:message>

  <wsdl:message name="skip">
    <wsdl:part name="skip" element="skip"/>
  </wsdl:message>
  <wsdl:message name="skipResponse">
    <wsdl:part name="skipResponse" element="skipResponse"/>
  </wsdl:message>

  <wsdl:message name="batchSkip">
    <wsdl:part name="batchSkip" element="batchSkip"/>
  </wsdl:message>
  <wsdl:message name="batchSkipResponse">
    <wsdl:part name="batchSkipResponse" element="batchSkipResponse"/>
  </wsdl:message>

  <wsdl:message name="forward">
    <wsdl:part name="forward" element="forward"/>
  </wsdl:message>
  <wsdl:message name="forwardResponse">
    <wsdl:part name="forwardResponse" element="forwardResponse"/>
  </wsdl:message>

  <wsdl:message name="batchForward">
    <wsdl:part name="batchForward" element="batchForward"/>
  </wsdl:message>
  <wsdl:message name="batchForwardResponse">
    <wsdl:part name="batchForwardResponse" element="batchForwardResponse"/>
  </wsdl:message>

  <wsdl:message name="delegate">
    <wsdl:part name="delegate" element="delegate"/>
  </wsdl:message>
  <wsdl:message name="delegateResponse">
    <wsdl:part name="delegateResponse" element="delegateResponse"/>
  </wsdl:message>

  <wsdl:message name="batchDelegate">
    <wsdl:part name="batchDelegate" element="batchDelegate"/>
  </wsdl:message>
  <wsdl:message name="batchDelegateResponse">
    <wsdl:part name="batchDelegateResponse" element="batchDelegateResponse"/>
  </wsdl:message>

  <wsdl:message name="getRendering">
    <wsdl:part name="getRendering" element="getRendering"/>
  </wsdl:message>
  <wsdl:message name="getRenderingResponse">
    <wsdl:part name="getRenderingResponse" element="getRenderingResponse"/>
  </wsdl:message>

  <wsdl:message name="getRenderingTypes">
    <wsdl:part name="getRenderingTypes" element="getRenderingTypes"/>
  </wsdl:message>
  <wsdl:message name="getRenderingTypesResponse">
    <wsdl:part name="getRenderingTypesResponse"
element="getRenderingTypesResponse"/>
  </wsdl:message>

  <wsdl:message name="getTaskDetails">
    <wsdl:part name="getTaskDetails" element="getTaskDetails"/>
  </wsdl:message>
  <wsdl:message name="getTaskDetailsResponse">
    <wsdl:part name="getTaskDetailsResponse"
element="getTaskDetailsResponse"/>
  </wsdl:message>

  <wsdl:message name="getTaskDescription">
    <wsdl:part name="getTaskDescription" element="getTaskDescription"/>
  </wsdl:message>
  <wsdl:message name="getTaskDescriptionResponse">
    <wsdl:part name="getTaskDescriptionResponse"
element="getTaskDescriptionResponse"/>
  </wsdl:message>

  <wsdl:message name="setOutput">
    <wsdl:part name="setOutput" element="setOutput"/>
  </wsdl:message>
  <wsdl:message name="setOutputResponse">
    <wsdl:part name="setOutputResponse" element="setOutputResponse"/>
  </wsdl:message>

  <wsdl:message name="deleteOutput">
    <wsdl:part name="deleteOutput" element="deleteOutput"/>
  </wsdl:message>
  <wsdl:message name="deleteOutputResponse">
    <wsdl:part name="deleteOutputResponse" element="deleteOutputResponse"/>
  </wsdl:message>

  <wsdl:message name="setFault">
    <wsdl:part name="setFault" element="setFault"/>
  </wsdl:message>
  <wsdl:message name="setFaultResponse">
    <wsdl:part name="setFaultResponse" element="setFaultResponse"/>
  </wsdl:message>

  <wsdl:message name="deleteFault">
    <wsdl:part name="deleteFault" element="deleteFault"/>
  </wsdl:message>
  <wsdl:message name="deleteFaultResponse">
    <wsdl:part name="deleteFaultResponse" element="deleteFaultResponse"/>
  </wsdl:message>

  <wsdl:message name="getInput">
    <wsdl:part name="getInput" element="getInput"/>
  </wsdl:message>
  <wsdl:message name="getInputResponse">
    <wsdl:part name="getInputResponse" element="getInputResponse"/>
  </wsdl:message>

  <wsdl:message name="getOutput">
    <wsdl:part name="getOutput" element="getOutput"/>
  </wsdl:message>
  <wsdl:message name="getOutputResponse">
    <wsdl:part name="getOutputResponse" element="getOutputResponse"/>
  </wsdl:message>

  <wsdl:message name="getFault">
    <wsdl:part name="getFault" element="getFault"/>
  </wsdl:message>
  <wsdl:message name="getFaultResponse">
    <wsdl:part name="getFaultResponse" element="getFaultResponse"/>
  </wsdl:message>

  <wsdl:message name="getMyTaskAbstracts">
    <wsdl:part name="getMyTaskAbstracts" element="getMyTaskAbstracts"/>
  </wsdl:message>
  <wsdl:message name="getMyTaskAbstractsResponse">
    <wsdl:part name="getMyTaskAbstractsResponse"
element="getMyTaskAbstractsResponse"/>
  </wsdl:message>

  <wsdl:message name="getMyTaskDetails">
    <wsdl:part name="getMyTaskDetails" element="getMyTaskDetails"/>
  </wsdl:message>
  <wsdl:message name="getMyTaskDetailsResponse">
    <wsdl:part name="getMyTaskDetailsResponse"
element="getMyTaskDetailsResponse"/>
  </wsdl:message>

  <wsdl:message name="query">
    <wsdl:part name="query" element="query"/>
  </wsdl:message>
  <wsdl:message name="queryResponse">
    <wsdl:part name="queryResponse" element="queryResponse"/>
  </wsdl:message>

  <wsdl:message name="activate">
    <wsdl:part name="activate" element="activate"/>
  </wsdl:message>
  <wsdl:message name="activateResponse">
    <wsdl:part name="activateResponse" element="activateResponse"/>
  </wsdl:message>

  <wsdl:message name="batchActivate">
    <wsdl:part name="batchActivate" element="batchActivate"/>
  </wsdl:message>
  <wsdl:message name="batchActivateResponse">
    <wsdl:part name="batchActivateResponse" element="batchActivateResponse"/>
  </wsdl:message>

  <wsdl:message name="nominate">
    <wsdl:part name="nominate" element="nominate"/>
  </wsdl:message>
  <wsdl:message name="nominateResponse">
    <wsdl:part name="nominateResponse" element="nominateResponse"/>
  </wsdl:message>

  <wsdl:message name="batchNominate">
    <wsdl:part name="batchNominate" element="batchNominate"/>
  </wsdl:message>
  <wsdl:message name="batchNominateResponse">
    <wsdl:part name="batchNominateResponse" element="batchNominateResponse"/>
  </wsdl:message>

  <wsdl:message name="setGenericHumanRole">
    <wsdl:part name="setGenericHumanRole" element="setGenericHumanRole"/>
  </wsdl:message>
  <wsdl:message name="setGenericHumanRoleResponse">
    <wsdl:part name="setGenericHumanRoleResponse"
element="setGenericHumanRoleResponse"/>
  </wsdl:message>

  <wsdl:message name="batchSetGenericHumanRole">
    <wsdl:part name="batchSetGenericHumanRole"
element="batchSetGenericHumanRole"/>
  </wsdl:message>
  <wsdl:message name="batchSetGenericHumanRoleResponse">
    <wsdl:part name="batchSetGenericHumanRoleResponse"
element="batchSetGenericHumanRoleResponse"/>
  </wsdl:message>

  <wsdl:message name="getOutcome">
    <wsdl:part name="getOutcome" element="getOutcome"/>
  </wsdl:message>
  <wsdl:message name="getOutcomeResponse">
    <wsdl:part name="getOutcomeResponse" element="getOutcomeResponse"/>
  </wsdl:message>

  <wsdl:message name="getTaskOperations">
    <wsdl:part name="getTaskOperations" element="getTaskOperations"/>
  </wsdl:message>
  <wsdl:message name="getTaskOperationsResponse">
    <wsdl:part name="getTaskOperationsResponse"
element="getTaskOperationsResponse"/>
  </wsdl:message>

  <wsdl:message name="getTaskInstanceData">
    <wsdl:part name="getTaskInstanceData" element="getTaskInstanceData"/>
  </wsdl:message>
  <wsdl:message name="getTaskInstanceDataResponse">
```

```
    <wsdl:part name="getTaskInstanceDataResponse"
element="getTaskInstanceDataResponse"/>
  </wsdl:message>

  <wsdl:message name="getTaskHistory">
    <wsdl:part name="getTaskHistory" element="getTaskHistory"/>
  </wsdl:message>
  <wsdl:message name="getTaskHistoryResponse">
    <wsdl:part name="getTaskHistoryResponse"
element="getTaskHistoryResponse"/>
  </wsdl:message>

  <wsdl:message name="setTaskStartDeadlineExpression">
    <wsdl:part name="setTaskStartDeadlineExpression"
element="setTaskStartDeadlineExpression"/>
  </wsdl:message>
  <wsdl:message name="setTaskStartDeadlineExpressionResponse">
    <wsdl:part name="setTaskStartDeadlineExpressionResponse"
element="setTaskStartDeadlineExpressionResponse"/>
  </wsdl:message>

  <wsdl:message name="setTaskStartDurationExpression">
    <wsdl:part name="setTaskStartDurationExpression"
element="setTaskStartDurationExpression"/>
  </wsdl:message>
  <wsdl:message name="setTaskStartDurationExpressionResponse">
    <wsdl:part name="setTaskStartDurationExpressionResponse"
element="setTaskStartDurationExpressionResponse"/>
  </wsdl:message>

  <wsdl:message name="setTaskCompletionDeadlineExpression">
    <wsdl:part name="setTaskCompletionDeadlineExpression"
element="setTaskCompletionDeadlineExpression"/>
  </wsdl:message>
  <wsdl:message name="setTaskCompletionDeadlineExpressionResponse">
    <wsdl:part name="setTaskCompletionDeadlineExpressionResponse"
element="setTaskCompletionDeadlineExpressionResponse"/>
  </wsdl:message>

  <wsdl:message name="setTaskCompletionDurationExpression">
    <wsdl:part name="setTaskCompletionDurationExpression"
element="setTaskCompletionDurationExpression"/>
  </wsdl:message>
  <wsdl:message name="setTaskCompletionDurationExpressionResponse">
    <wsdl:part name="setTaskCompletionDurationExpressionResponse"
element="setTaskCompletionDurationExpressionResponse"/>
  </wsdl:message>

  <!-- Declaration of fault messages -->
  <wsdl:message name="illegalStateFault">
    <wsdl:part name="illegalState" element="illegalState"/>
  </wsdl:message>
  <wsdl:message name="illegalArgumentFault">
    <wsdl:part name="illegalArgument" element="illegalArgument"/>
  </wsdl:message>
  <wsdl:message name="illegalAccessFault">
    <wsdl:part name="illegalAccess" element="illegalAccess"/>
  </wsdl:message>
  <wsdl:message name="illegalOperationFault">
    <wsdl:part name="illegalOperation" element="illegalOperation"/>
  </wsdl:message>
  <wsdl:message name="recipientNotAllowed">
    <wsdl:part name="recipientNotAllowed" element="recipientNotAllowed"/>
  </wsdl:message>

  <!-- Port type definition -->
  <wsdl:portType name="taskOperations">

    <wsdl:operation name="claim">
      <wsdl:input message="claim"/>
      <wsdl:output message="claimResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="batchClaim">
      <wsdl:input message="batchClaim"/>
      <wsdl:output message="batchClaimResponse"/>
    </wsdl:operation>

    <wsdl:operation name="start">
      <wsdl:input message="start"/>
      <wsdl:output message="startResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="batchStart">
      <wsdl:input message="batchStart"/>
      <wsdl:output message="batchStartResponse"/>
    </wsdl:operation>

    <wsdl:operation name="stop">
      <wsdl:input message="stop"/>
      <wsdl:output message="stopResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="batchStop">
      <wsdl:input message="batchStop"/>
      <wsdl:output message="batchStopResponse"/>
    </wsdl:operation>

    <wsdl:operation name="release">
      <wsdl:input message="release"/>
      <wsdl:output message="releaseResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="batchRelease">
      <wsdl:input message="batchRelease"/>
      <wsdl:output message="batchReleaseResponse"/>
    </wsdl:operation>

    <wsdl:operation name="suspend">
      <wsdl:input message="suspend"/>
      <wsdl:output message="suspendResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="batchSuspend">
      <wsdl:input message="batchSuspend"/>
      <wsdl:output message="batchSuspendResponse"/>
    </wsdl:operation>

    <wsdl:operation name="suspendUntil">
      <wsdl:input message="suspendUntil"/>
      <wsdl:output message="suspendUntilResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="batchSuspendUntil">
      <wsdl:input message="batchSuspendUntil"/>
      <wsdl:output message="batchSuspendUntilResponse"/>
    </wsdl:operation>

    <wsdl:operation name="resume">
      <wsdl:input message="resume"/>
      <wsdl:output message="resumeResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="batchResume">
      <wsdl:input message="batchResume"/>
      <wsdl:output message="batchResumeResponse"/>
    </wsdl:operation>

    <wsdl:operation name="complete">
      <wsdl:input message="complete"/>
      <wsdl:output message="completeResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="batchComplete">
      <wsdl:input message="batchComplete"/>
      <wsdl:output message="batchCompleteResponse"/>
    </wsdl:operation>

    <wsdl:operation name="remove">
      <wsdl:input message="remove"/>
      <wsdl:output message="removeResponse"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="batchRemove">
      <wsdl:input message="batchRemove"/>
      <wsdl:output message="batchRemoveResponse"/>
    </wsdl:operation>

    <wsdl:operation name="fail">
      <wsdl:input message="fail"/>
      <wsdl:output message="failResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="batchFail">
      <wsdl:input message="batchFail"/>
      <wsdl:output message="batchFailResponse"/>
    </wsdl:operation>

    <wsdl:operation name="setPriority">
      <wsdl:input message="setPriority"/>
      <wsdl:output message="setPriorityResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="batchSetPriority">
      <wsdl:input message="batchSetPriority"/>
      <wsdl:output message="batchSetPriorityResponse"/>
    </wsdl:operation>

    <wsdl:operation name="addAttachment">
      <wsdl:input message="addAttachment"/>
      <wsdl:output message="addAttachmentResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="getAttachmentInfos">
      <wsdl:input message="getAttachmentInfos"/>
      <wsdl:output message="getAttachmentInfosResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="getAttachment">
      <wsdl:input message="getAttachment"/>
      <wsdl:output message="getAttachmentResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="deleteAttachment">
      <wsdl:input message="deleteAttachment"/>
      <wsdl:output message="deleteAttachmentResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="addComment">
      <wsdl:input message="addComment"/>
      <wsdl:output message="addCommentResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="getComments">
      <wsdl:input message="getComments"/>
      <wsdl:output message="getCommentsResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="skip">
      <wsdl:input message="skip"/>
      <wsdl:output message="skipResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="batchSkip">
      <wsdl:input message="batchSkip"/>
      <wsdl:output message="batchSkipResponse"/>
    </wsdl:operation>

    <wsdl:operation name="forward">
      <wsdl:input message="forward"/>
      <wsdl:output message="forwardResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="batchForward">
      <wsdl:input message="batchForward"/>
      <wsdl:output message="batchForwardResponse"/>
    </wsdl:operation>

    <wsdl:operation name="delegate">
      <wsdl:input message="delegate"/>
      <wsdl:output message="delegateResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
```

```
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault" 
message="illegalOperationFault"/>
      <wsdl:fault name="recipientNotAllowed" message="recipientNotAllowed"/>
    </wsdl:operation>

    <wsdl:operation name="batchDelegate">
      <wsdl:input message="batchDelegate"/>
      <wsdl:output message="batchDelegateResponse"/>
    </wsdl:operation>

    <wsdl:operation name="getRendering">
      <wsdl:input message="getRendering"/>
      <wsdl:output message="getRenderingResponse"/>
      <wsdl:fault name="illegalArgumentFault" 
message="illegalArgumentFault"/>
    </wsdl:operation>

    <wsdl:operation name="getRenderingTypes">
      <wsdl:input message="getRenderingTypes"/>
      <wsdl:output message="getRenderingTypesResponse"/>
      <wsdl:fault name="illegalArgumentFault" 
message="illegalArgumentFault"/>
    </wsdl:operation>

    <wsdl:operation name="getTaskDetails">
      <wsdl:input message="getTaskDetails"/>
      <wsdl:output message="getTaskDetailsResponse"/>
      <wsdl:fault name="illegalArgumentFault" 
message="illegalArgumentFault"/>
    </wsdl:operation>

    <wsdl:operation name="getTaskDescription">
      <wsdl:input message="getTaskDescription"/>
      <wsdl:output message="getTaskDescriptionResponse"/>
      <wsdl:fault name="illegalArgumentFault" 
message="illegalArgumentFault"/>
    </wsdl:operation>

    <wsdl:operation name="setOutput">
      <wsdl:input message="setOutput"/>
      <wsdl:output message="setOutputResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault" 
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault" 
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="deleteOutput">
      <wsdl:input message="deleteOutput"/>
      <wsdl:output message="deleteOutputResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault" 
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="setFault">
      <wsdl:input message="setFault"/>
      <wsdl:output message="setFaultResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="deleteFault">
      <wsdl:input message="deleteFault"/>
      <wsdl:output message="deleteFaultResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="getInput">
      <wsdl:input message="getInput"/>
      <wsdl:output message="getInputResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="getOutput">
      <wsdl:input message="getOutput"/>
      <wsdl:output message="getOutputResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="getFault">
      <wsdl:input message="getFault"/>
      <wsdl:output message="getFaultResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="getMyTaskAbstracts">
      <wsdl:input message="getMyTaskAbstracts"/>
      <wsdl:output message="getMyTaskAbstractsResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="getMyTaskDetails">
      <wsdl:input message="getMyTaskDetails"/>
      <wsdl:output message="getMyTaskDetailsResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="query">
      <wsdl:input message="query"/>
      <wsdl:output message="queryResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
    </wsdl:operation>

    <wsdl:operation name="activate">
      <wsdl:input message="activate"/>
      <wsdl:output message="activateResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="batchActivate">
      <wsdl:input message="batchActivate"/>
      <wsdl:output message="batchActivateResponse"/>
    </wsdl:operation>

    <wsdl:operation name="nominate">
      <wsdl:input message="nominate"/>
      <wsdl:output message="nominateResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="batchNominate">
      <wsdl:input message="batchNominate"/>
      <wsdl:output message="batchNominateResponse"/>
    </wsdl:operation>

    <wsdl:operation name="setGenericHumanRole">
      <wsdl:input message="setGenericHumanRole"/>
      <wsdl:output message="setGenericHumanRoleResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="batchSetGenericHumanRole">
      <wsdl:input message="batchSetGenericHumanRole"/>
      <wsdl:output message="batchSetGenericHumanRoleResponse"/>
    </wsdl:operation>

    <wsdl:operation name="getOutcome">
      <wsdl:input message="getOutcome"/>
      <wsdl:output message="getOutcomeResponse"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="getTaskOperations">
      <wsdl:input message="getTaskOperations"/>
      <wsdl:output message="getTaskOperationsResponse"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="getTaskInstanceData">
      <wsdl:input message="getTaskInstanceData"/>
      <wsdl:output message="getTaskInstanceDataResponse"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="getTaskHistory">
      <wsdl:input message="getTaskHistory"/>
      <wsdl:output message="getTaskHistoryResponse"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="setTaskStartDeadlineExpression">
```

```
      <wsdl:input message="setTaskStartDeadlineExpression"/>
      <wsdl:output message="setTaskStartDeadlineExpressionResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="setTaskStartDurationExpression">
      <wsdl:input message="setTaskStartDurationExpression"/>
      <wsdl:output message="setTaskStartDurationExpressionResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="setTaskCompletionDeadlineExpression">
      <wsdl:input message="setTaskCompletionDeadlineExpression"/>
      <wsdl:output message="setTaskCompletionDeadlineExpressionResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

    <wsdl:operation name="setTaskCompletionDurationExpression">
      <wsdl:input message="setTaskCompletionDurationExpression"/>
      <wsdl:output message="setTaskCompletionDurationExpressionResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
      <wsdl:fault name="illegalOperationFault"
message="illegalOperationFault"/>
    </wsdl:operation>

  </wsdl:portType>
</wsdl:definitions>
```

E. WS-HumanTask Parent API Port Type

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Copyright (c) OASIS Open 2009. All Rights Reserved.
-->
<wsdl:definitions
  targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
humantask/leantask/api/200803"
  xmlns="http://docs.oasis-open.org/ns/bpel4people/ws-
humantask/leantask/api/200803"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:htd="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803"
  xmlns:htt="http://docs.oasis-open.org/ns/bpel4people/ws-
humantask/types/200803">

  <wsdl:documentation>
    Web Service Definition for WS-HumanTask 1.1 - Operations for Task Parent
Applications
  </wsdl:documentation>

  <wsdl:types>
    <xsd:schema
      targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
humantask/leantask/api/200803"
      elementFormDefault="qualified"
      blockDefault="#all">

      <xsd:import
        namespace="http://docs.oasis-open.org/ns/bpel4people/ws-
humantask/200803"
        schemaLocation="ws-humantask.xsd"/>
      <xsd:import
        namespace="http://docs.oasis-open.org/ns/bpel4people/ws-
humantask/types/200803"
        schemaLocation="ws-humantask-types.xsd"/>

      <!-- Input and output elements -->
      <xsd:element name="registerLeanTaskDefinition">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="taskDefinition" type="htd:tLeanTask" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="registerLeanTaskDefinitionResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="taskName" type="xsd:NCName" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

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

```
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="taskName" type="xsd:NCName" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="unregisterLeanTaskDefinitionResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="taskName" type="xsd:NCName" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="listLeanTaskDefinitions">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:annotation>
              <xsd:documentation>Empty message</xsd:documentation>
            </xsd:annotation>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="listLeanTaskDefinitionsResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="leanTaskDefinitions">
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element name="leanTaskDefinition" type="htd:tLeanTask"
minOccurs="0" maxOccurs="unbounded" />
                </xsd:sequence>
              </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="createLeanTask">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="inputMessage">
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:any processContents="lax" namespace="##any" />
                </xsd:sequence>
              </xsd:complexType>
            </xsd:element>
            <xsd:element name="taskDefinition" type="htd:tLeanTask"
minOccurs="0"/>
            <xsd:element name="taskName" type="xsd:NCName" minOccurs="0" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="createLeanTaskResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="outputMessage">
```

```
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:any processContents="lax" namespace="##any" />
                </xsd:sequence>
              </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="createLeanTaskAsync">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="inputMessage">
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:any processContents="lax" namespace="##any" />
                </xsd:sequence>
              </xsd:complexType>
            </xsd:element>
            <xsd:element name="taskDefinition" type="htd:tLeanTask"
minOccurs="0"/>
            <xsd:element name="taskName" type="xsd:NCName" minOccurs="0" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="createLeanTaskAsyncResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:annotation>
              <xsd:documentation>Empty message</xsd:documentation>
            </xsd:annotation>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="createLeanTaskAsyncCallback">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="outputMessage">
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:any processContents="lax" namespace="##any" />
                </xsd:sequence>
              </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <!-- Fault elements -->
      <xsd:element name="illegalState">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="status" type="htt:tStatus"/>
            <xsd:element name="message" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="illegalArgument" type="xsd:string"/>

      <xsd:element name="illegalAccess" type="xsd:string"/>

    </xsd:schema>
  </wsdl:types>

  <!-- Declaration of messages -->
  <wsdl:message name="registerLeanTaskDefinition">
    <wsdl:part name="registerLeanTaskDefinition"
element="registerLeanTaskDefinition"/>
  </wsdl:message>
  <wsdl:message name="registerLeanTaskDefinitionResponse">
    <wsdl:part name="registerLeanTaskDefinitionResponse"
element="registerLeanTaskDefinitionResponse"/>
  </wsdl:message>

  <wsdl:message name="unregisterLeanTaskDefinition">
    <wsdl:part name="unregisterLeanTaskDefinition"
element="unregisterLeanTaskDefinition"/>
  </wsdl:message>
  <wsdl:message name="unregisterLeanTaskDefinitionResponse">
    <wsdl:part name="unregisterLeanTaskDefinitionResponse"
element="unregisterLeanTaskDefinitionResponse"/>
  </wsdl:message>

  <wsdl:message name="listLeanTaskDefinitions">
    <wsdl:part name="listLeanTaskDefinitions"
element="listLeanTaskDefinitions"/>
  </wsdl:message>
  <wsdl:message name="listLeanTaskDefinitionsResponse">
    <wsdl:part name="listLeanTaskDefinitionsResponse"
element="listLeanTaskDefinitionsResponse"/>
  </wsdl:message>

  <wsdl:message name="createLeanTask">
    <wsdl:part name="createLeanTask" element="createLeanTask"/>
  </wsdl:message>
  <wsdl:message name="createLeanTaskResponse">
    <wsdl:part name="createLeanTaskResponse"
element="createLeanTaskResponse"/>
  </wsdl:message>

  <wsdl:message name="createLeanTaskAsync">
    <wsdl:part name="createLeanTaskAsync" element="createLeanTaskAsync"/>
  </wsdl:message>
  <wsdl:message name="createLeanTaskAsyncResponse">
    <wsdl:part name="createLeanTaskAsyncResponse"
element="createLeanTaskAsyncResponse"/>
  </wsdl:message>

  <wsdl:message name="createLeanTaskAsyncCallback">
    <wsdl:part name="createLeanTaskAsyncCallback"
element="createLeanTaskAsyncCallback"/>
  </wsdl:message>

  <!-- Declaration of fault messages -->
  <wsdl:message name="illegalStateFault">
    <wsdl:part name="illegalState" element="illegalState"/>
  </wsdl:message>
  <wsdl:message name="illegalArgumentFault">
    <wsdl:part name="illegalArgument" element="illegalArgument"/>
  </wsdl:message>
  <wsdl:message name="illegalAccessFault">
    <wsdl:part name="illegalAccess" element="illegalAccess"/>
  </wsdl:message>

  <!-- Port type definitions -->
  <wsdl:portType name="leanTaskOperations">

    <wsdl:operation name="registerLeanTaskDefinition">
      <wsdl:input message="registerLeanTaskDefinition"/>
      <wsdl:output message="registerLeanTaskDefinitionResponse"/>
      <wsdl:fault name="illegalStateFault" message="illegalStateFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
    </wsdl:operation>

    <wsdl:operation name="unregisterLeanTaskDefinition">
      <wsdl:input message="unregisterLeanTaskDefinition"/>
      <wsdl:output message="unregisterLeanTaskDefinitionResponse"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
    </wsdl:operation>

    <wsdl:operation name="listLeanTaskDefinitions">
      <wsdl:input message="listLeanTaskDefinitions"/>
      <wsdl:output message="listLeanTaskDefinitionsResponse"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
    </wsdl:operation>

    <wsdl:operation name="createLeanTask">
      <wsdl:input message="createLeanTask"/>
      <wsdl:output message="createLeanTaskResponse"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
    </wsdl:operation>

    <wsdl:operation name="createLeanTaskAsync">
      <wsdl:input message="createLeanTaskAsync"/>
      <wsdl:output message="createLeanTaskAsyncResponse"/>
      <wsdl:fault name="illegalArgumentFault"
message="illegalArgumentFault"/>
      <wsdl:fault name="illegalAccessFault" message="illegalAccessFault"/>
    </wsdl:operation>

  </wsdl:portType>

  <wsdl:portType name="leanTaskCallbackOperations">

    <wsdl:operation name="createLeanTaskAsyncCallback">
      <wsdl:input message="createLeanTaskAsyncCallback"/>
    </wsdl:operation>

  </wsdl:portType>

</wsdl:definitions>
```

F. WS-HumanTask Protocol Handler Port Types

```
 <?xml version="1.0" encoding="UTF-8"?>
<!--
  Copyright (c) OASIS Open 2009. All Rights Reserved.
-->
<wsdl:definitions
  targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
humantask/protocol/200803"
  xmlns="http://docs.oasis-open.org/ns/bpel4people/ws-
humantask/protocol/200803"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:htp="http://docs.oasis-open.org/ns/bpel4people/ws-
humantask/protocol/200803">

  <wsdl:documentation>
    Web Service Definition for WS-HumanTask 1.1 - Operations WS-HumanTask
Protocol Participants
  </wsdl:documentation>

  <wsdl:types>
  <xsd:schema
    targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
humantask/protocol/200803"
    elementFormDefault="qualified"
    blockDefault="#all">

    <xsd:complexType name="tProtocolMsgType">
    <xsd:sequence>
      <xsd:any namespace="##other" processContents="lax" minOccurs="0"
      maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:anyAttribute namespace="##any" processContents="lax" />
    </xsd:complexType>

    <xsd:element name="skipped" type="htp:tProtocolMsgType" />
    <xsd:element name="fault" type="htp:tProtocolMsgType" />
    <xsd:element name="exit" type="htp:tProtocolMsgType" />

    <xsd:element name="responseAction" type="xsd:anyURI" />
    <xsd:element name="responseOperation" type="xsd:NCName" />

    </xsd:schema>
  </wsdl:types>

  <wsdl:message name="skipped">
    <wsdl:part name="parameters" element="skipped" />
  </wsdl:message>
  <wsdl:message name="fault">
    <wsdl:part name="parameters" element="fault" />
  </wsdl:message>
  <wsdl:message name="exit">
    <wsdl:part name="parameters" element="exit" />
  </wsdl:message>

  <wsdl:portType name="clientParticipantPortType">
    <wsdl:operation name="skippedOperation">
      <wsdl:input message="skipped" />
    </wsdl:operation>
    <wsdl:operation name="faultOperation">
      <wsdl:input message="fault" />
    </wsdl:operation>
  </wsdl:portType>

  <wsdl:portType name="humanTaskParticipantPortType">
    <wsdl:operation name="exitOperation">
      <wsdl:input message="exit" />
    </wsdl:operation>
  </wsdl:portType>

</wsdl:definitions>
```

G. WS-HumanTask Context Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Copyright (c) OASIS Open 2009. All Rights Reserved.
-->
<xsd:schema
  targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
humantask/context/200803"
  xmlns="http://docs.oasis-open.org/ns/bpel4people/ws-
humantask/context/200803"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:htt="http://docs.oasis-open.org/ns/bpel4people/ws-
humantask/types/200803"
  elementFormDefault="qualified"
  blockDefault="#all">

  <xsd:annotation>
    <xsd:documentation>
      XML Schema for WS-HumanTask 1.1 - Human Task Context for Task
Interactions
    </xsd:documentation>
  </xsd:annotation>

  <!-- other namespaces -->
  <xsd:import
    namespace="http://www.w3.org/XML/1998/namespace"
    schemaLocation="http://www.w3.org/2001/xml.xsd"/>
  <xsd:import
    namespace="http://docs.oasis-open.org/ns/bpel4people/ws-
humantask/types/200803"
    schemaLocation="ws-humantask-types.xsd"/>

  <!-- human task context -->
  <xsd:element name="humanTaskRequestContext"
type="tHumanTaskRequestContext"/>
  <xsd:complexType name="tHumanTaskRequestContext">
    <xsd:complexContent>
      <xsd:extension base="tHumanTaskContextBase">
        <xsd:sequence>
          <xsd:element name="peopleAssignments" type="tPeopleAssignments"
minOccurs="0"/>
          <xsd:element name="isSkipable" type="xsd:boolean" minOccurs="0"/>
          <xsd:element name="expirationTime" type="xsd:dateTime"
minOccurs="0"/>
          <xsd:element name="activationDeferralTime" type="xsd:dateTime"
minOccurs="0"/>
          <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name="humanTaskResponseContext"
type="tHumanTaskResponseContext"/>
```

```
  <xsd:complexType name="tHumanTaskResponseContext">
    <xsd:complexContent>
      <xsd:extension base="tHumanTaskContextBase">
        <xsd:sequence>
          <xsd:element name="actualOwner" type="htt:tUser"/>
          <xsd:element name="actualPeopleAssignments"
type="tPeopleAssignments"/>
          <xsd:element name="outcome" type="xsd:string" minOccurs="0"/>
          <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="tHumanTaskContextBase" abstract="true">
    <xsd:sequence>
      <xsd:element name="priority" type="htt:tPriority" minOccurs="0"/>
      <xsd:element name="attachments" type="tAttachments" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>

  <!-- people assignments -->
  <xsd:complexType name="tPeopleAssignments">
    <xsd:sequence>
      <xsd:element ref="genericHumanRole" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="genericHumanRole" type="tGenericHumanRole"
abstract="true" block="restriction extension"/>
  <xsd:element name="potentialOwners" type="tGenericHumanRole"
substitutionGroup="genericHumanRole"/>
  <xsd:element name="excludedOwners" type="tGenericHumanRole"
substitutionGroup="genericHumanRole"/>
  <xsd:element name="taskInitiator" type="tGenericHumanRole"
substitutionGroup="genericHumanRole"/>
  <xsd:element name="taskStakeholders" type="tGenericHumanRole"
substitutionGroup="genericHumanRole"/>
  <xsd:element name="businessAdministrators" type="tGenericHumanRole"
substitutionGroup="genericHumanRole"/>
  <xsd:element name="recipients" type="tGenericHumanRole"
substitutionGroup="genericHumanRole"/>
  <xsd:complexType name="tGenericHumanRole">
    <xsd:sequence>
      <xsd:element ref="htt:organizationalEntity"/>
    </xsd:sequence>
  </xsd:complexType>

  <!-- attachments -->
  <xsd:complexType name="tAttachments">
    <xsd:sequence>
      <xsd:element name="returnAttachments" type="tReturnAttachments"
minOccurs="0"/>
      <xsd:element ref="htt:attachment" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:simpleType name="tReturnAttachments">
    <xsd:restriction base="xsd:string">
```

```
      <xsd:enumeration value="all"/>
      <xsd:enumeration value="newOnly"/>
      <xsd:enumeration value="none"/>
    </xsd:restriction>
  </xsd:simpleType>

</xsd:schema>
```

H. WS-HumanTask Policy Assertion Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Copyright (c) OASIS Open 2009. All Rights Reserved.
-->
<xsd:schema
  targetNamespace="http://docs.oasis-open.org/ns/bpel4people/ws-
humantask/policy/200803"
  xmlns="http://docs.oasis-open.org/ns/bpel4people/ws-
humantask/policy/200803"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:wsp="http://www.w3.org/ns/ws-policy"
  elementFormDefault="qualified"
  blockDefault="#all">

  <xsd:annotation>
    <xsd:documentation>
      XML Schema for WS-HumanTask 1.1 - WS-HumanTask Policy Assertion
    </xsd:documentation>
  </xsd:annotation>

  <!-- other namespaces -->
  <xsd:import
      namespace="http://www.w3.org/ns/ws-policy"
      schemaLocation="http://www.w3.org/2007/02/ws-policy.xsd" />

  <!-- ws-humantask policy assertion -->
  <xsd:element name="HumanTaskAssertion" type="tHumanTaskAssertion"/>
  <xsd:complexType name="tHumanTaskAssertion" >
    <xsd:attribute ref="wsp:Optional" />
    <xsd:anyAttribute namespace="##any" processContents="lax" />
  </xsd:complexType>

</xsd:schema>
```

I. Sample

This appendix contains the full sample used in this specification.

WSDL Definition

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Copyright (c) OASIS Open 2009. All Rights Reserved.
-->
<wsdl:definitions name="ClaimApproval"
  targetNamespace="http://www.example.com/claims"
  xmlns:tns="http://www.example.com/claims"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <wsdl:documentation>
    Example for WS-HumanTask 1.1 - WS-HumanTask Task Interface Definition
  </wsdl:documentation>

  <wsdl:types>
    <xsd:schema
      targetNamespace="http://www.example.com/claims"
      xmlns:tns="http://www.example.com/claims"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      elementFormDefault="qualified">
      <xsd:element name="ClaimApprovalData">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="cust">
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element name="id" type="xsd:string">
                  </xsd:element>
                  <xsd:element name="firstname" type="xsd:string">
                  </xsd:element>
                  <xsd:element name="lastname" type="xsd:string">
                  </xsd:element>
                </xsd:sequence>
              </xsd:complexType>
            </xsd:element>
            <xsd:element name="amount" type="xsd:double" />
            <xsd:element name="region" type="xsd:string" />
            <xsd:element name="prio" type="xsd:int" />
            <xsd:element name="activateAt" type="xsd:dateTime" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:schema>
  </wsdl:types>

  <wsdl:message name="ClaimApprovalRequest">
    <wsdl:part name="ClaimApprovalRequest"
      element="tns:ClaimApprovalData" />
  </wsdl:message>
  <wsdl:message name="ClaimApprovalResponse">
    <wsdl:part name="ClaimApprovalResponse" type="xsd:boolean" />
  </wsdl:message>
  <wsdl:message name="notifyRequest">
    <wsdl:part name="firstname" type="xsd:string" />
    <wsdl:part name="lastname" type="xsd:string" />
  </wsdl:message>

  <wsdl:portType name="ClaimsHandlingPT">
    <wsdl:operation name="approve">
      <wsdl:input message="tns:ClaimApprovalRequest" />
    </wsdl:operation>
    <wsdl:operation name="escalate">
      <wsdl:input message="tns:ClaimApprovalRequest" />
    </wsdl:operation>
  </wsdl:portType>

  <wsdl:portType name="ClaimsHandlingCallbackPT">
    <wsdl:operation name="approvalResponse">
      <wsdl:input message="tns:ClaimApprovalResponse" />
    </wsdl:operation>
  </wsdl:portType>

  <wsdl:portType name="ClaimApprovalReminderPT">
    <wsdl:operation name="notify">
      <wsdl:input message="tns:notifyRequest" />
    </wsdl:operation>
  </wsdl:portType>

</wsdl:definitions>
```

Human Interaction Definition

```
 <?xml version="1.0" encoding="UTF-8"?>
<!--
  Copyright (c) OASIS Open 2009. All Rights Reserved.
-->
<htd:humanInteractions
  xmlns:htd="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803"
  xmlns:htt="http://docs.oasis-open.org/ns/bpel4people/ws-
humantask/types/200803"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:cl="http://www.example.com/claims/"
  xmlns:tns="http://www.example.com"
  targetNamespace="http://www.example.com"
  xsi:schemaLocation="http://docs.oasis-open.org/ns/bpel4people/ws-
humantask/200803 ../../xml/ws-humantask.xsd">

  <htd:documentation>
    Example for WS-HumanTask 1.1 - WS-HumanTask Task Definition
  </htd:documentation>

  <htd:import importType="http://schemas.xmlsoap.org/wsdl/"
    location="ws-humantask-example-claim-approval.wsdl"
    namespace="http://www.example.com/claims/" />

  <htd:logicalPeopleGroups>

    <htd:logicalPeopleGroup name="regionalClerks">
      <htd:documentation xml:lang="en-US">
        The group of clerks responsible for the region specified.
      </htd:documentation>
      <htd:parameter name="region" type="xsd:string" />
    </htd:logicalPeopleGroup>

    <htd:logicalPeopleGroup name="regionalManager">
      <htd:documentation xml:lang="en-US">
        The manager responsible for the region specified.
      </htd:documentation>
      <htd:parameter name="region" type="xsd:string" />
    </htd:logicalPeopleGroup>

    <htd:logicalPeopleGroup name="clerksManager">
      <htd:documentation xml:lang="en-US">
        The manager of the clerk whose user ID is passed as parameter.
      </htd:documentation>
      <htd:parameter name="clerkUserID" type="xsd:string" />
    </htd:logicalPeopleGroup>

    <htd:logicalPeopleGroup name="directorClaims">
      <htd:documentation xml:lang="en-US">
        The functional director responsible for claims processing.
      </htd:documentation>
    </htd:logicalPeopleGroup>

  </htd:logicalPeopleGroups>

  <htd:tasks>

    <htd:task name="ApproveClaim">
      <htd:documentation xml:lang="en-US">
        This task is used to handle claims that require manual
        approval.
      </htd:documentation>

      <htd:interface portType="cl:ClaimsHandlingPT"
        operation="approve"
        responsePortType="cl:ClaimsHandlingCallbackPT"
        responseOperation="approvalResponse" />

      <htd:priority>
        htd:getInput("ClaimApprovalRequest")/prio
      </htd:priority>

      <htd:peopleAssignments>
        <htd:potentialOwners>
          <htd:from logicalPeopleGroup="regionalClerks">
            <htd:argument name="region">
              htd:getInput("ClaimApprovalRequest")/region
            </htd:argument>
          </htd:from>
        </htd:potentialOwners>

        <htd:businessAdministrators>
```

```
          <htd:from logicalPeopleGroup="regionalManager">
            <htd:argument name="region">
              htd:getInput("ClaimApprovalRequest")/region
            </htd:argument>
          </htd:from>
        </htd:businessAdministrators>
      </htd:peopleAssignments>

      <htd:delegation potentialDelegatees="nobody" />

      <htd:presentationElements>

        <htd:name xml:lang="en-US">Approve Claim</htd:name>
        <htd:name xml:lang="de-DE">
          Genehmigung der Schadensforderung
        </htd:name>

        <htd:presentationParameters>
          <htd:presentationParameter name="firstname"
            type="xsd:string">
            htd:getInput("ClaimApprovalRequest")/cust/firstname
          </htd:presentationParameter>
          <htd:presentationParameter name="lastname"
            type="xsd:string">
            htd:getInput("ClaimApprovalRequest")/cust/lastname
          </htd:presentationParameter>
          <htd:presentationParameter name="euroAmount"
            type="xsd:double">
            htd:getInput("ClaimApprovalRequest")/amount
          </htd:presentationParameter>
        </htd:presentationParameters>

        <htd:subject xml:lang="en-US">
          Approve the insurance claim for €$euroAmount$ on behalf of
          $firstname$ $lastname$
        </htd:subject>
        <htd:subject xml:lang="de-DE">
          Genehmigung der Schadensforderung über €$euroAmount$ für
          $firstname$ $lastname$
        </htd:subject>

        <htd:description xml:lang="en-US" contentType="text/plain">
          Approve this claim following corporate guideline
          #4711.0815/7 ...
        </htd:description>
        <htd:description xml:lang="en-US" contentType="text/html">
          <![CDATA[
          <p>
            Approve this claim following corporate guideline
            <b>#4711.0815/7</b>
            ...
          </p>
          ]]>
        </htd:description>
        <htd:description xml:lang="de-DE" contentType="text/plain">
          Genehmigen Sie diese Schadensforderung entsprechend
          Richtlinie Nr. 4711.0815/7 ...
        </htd:description>
        <htd:description xml:lang="de-DE" contentType="text/html">
          <![CDATA[
          <p>
            Genehmigen Sie diese Schadensforderung entsprechend
            Richtlinie
            <b>Nr. 4711.0815/7</b>
            ...
          </p>
          ]]>
        </htd:description>

      </htd:presentationElements>


      <htd:deadlines>

        <htd:startDeadline name="sendReminder">
          <htd:documentation xml:lang="en-US">
            If not started within 3 days, - escalation notifications
            are sent if the claimed amount is less than 10000 - to the
            task's potential owners to remind them or their todo - to
            the regional manager, if this approval is of high priority
            (0,1, or 2) - the task is reassigned to Alan if the
            claimed amount is greater than or equal 10000
          </htd:documentation>
          <htd:for>P3D</htd:for>

          <htd:escalation name="reminder">

            <htd:condition>
              <![CDATA[
                htd:getInput("ClaimApprovalRequest")/amount < 10000
              ]]>
            </htd:condition>

            <htd:toParts>
              <htd:toPart name="firstname">
                htd:getInput("ClaimApprovalRequest","ApproveClaim")
                /firstname
              </htd:toPart>
              <htd:toPart name="lastname">
                htd:getInput("ClaimApprovalRequest","ApproveClaim")
                /lastname
              </htd:toPart>
            </htd:toParts>

            <htd:localNotification
              reference="tns:ClaimApprovalReminder">

              <htd:documentation xml:lang="en-US">
                Reuse the predefined notification
                "ClaimApprovalReminder". Overwrite the recipients with
                the task's potential owners.
              </htd:documentation>

              <htd:peopleAssignments>
                <htd:recipients>
                  <htd:from>
```

```
                        htd:getPotentialOwners("ApproveClaim")
                      </htd:from>
                    </htd:recipients>
                  </htd:peopleAssignments>

                </htd:localNotification>

              </htd:escalation>

              <htd:escalation name="highPrio">

                <htd:condition>
                  <![CDATA[
                    (htd:getInput("ClaimApprovalRequest")/amount < 10000
                  && htd:getInput("ClaimApprovalRequest")/prio <= 2)
                  ]]>
                </htd:condition>

                <!-- task input implicitly passed to the notification -->

                <htd:notification name="ClaimApprovalOverdue">
                  <htd:documentation xml:lang="en-US">
                    An inline defined notification using the approval data
                    as its input.
                  </htd:documentation>

                  <htd:interface portType="cl:ClaimsHandlingPT"
                    operation="escalate" />

                  <htd:peopleAssignments>
                    <htd:recipients>
                      <htd:from logicalPeopleGroup="regionalManager">
                        <htd:argument name="region">
                          htd:getInput("ClaimApprovalRequest")/region
                        </htd:argument>
                      </htd:from>
                    </htd:recipients>
                  </htd:peopleAssignments>

                  <htd:presentationElements>
                    <htd:name xml:lang="en-US">
                      Claim approval overdue
                    </htd:name>
                    <htd:name xml:lang="de-DE">
                      Überfällige Schadensforderungsgenehmigung
                    </htd:name>
                  </htd:presentationElements>

                </htd:notification>

              </htd:escalation>

              <htd:escalation name="highAmountReassign">

                <htd:condition>
                  <![CDATA[
                    htd:getInput("ClaimApprovalRequest")/amount >= 10000
                  ]]>
```

```
          </htd:condition>

          <htd:reassignment>
            <htd:documentation>
              Reassign task to Alan if amount is greater than or
              equal 10000.
            </htd:documentation>

            <htd:potentialOwners>
              <htd:from>
                <htd:literal>
                  <htt:organizationalEntity>
                    <htt:user>Alan</htt:user>
                  </htt:organizationalEntity>
                </htd:literal>
              </htd:from>
            </htd:potentialOwners>

          </htd:reassignment>

        </htd:escalation>

      </htd:startDeadline>


      <htd:completionDeadline name="notifyManager">
        <htd:documentation xml:lang="en-US">
          When not completed within 3 hours after having been
          claimed, the manager of the clerk who claimed the activity
          is notified.
        </htd:documentation>
        <htd:for>PT3H</htd:for>

        <htd:escalation name="delayedApproval">

          <htd:notification name="ClaimApprovalOverdue">
            <htd:documentation xml:lang="en-US">
              An inline defined notification using the approval data
              as its input.
            </htd:documentation>

            <htd:interface portType="cl:ClaimsHandlingPT"
              operation="escalate" />

            <htd:peopleAssignments>
              <htd:recipients>
                <htd:from logicalPeopleGroup="clerksManager">
                  <htd:argument name="clerkUserID">
                    htd:getActualOwner("ApproveClaim")
                  </htd:argument>
                </htd:from>
              </htd:recipients>
            </htd:peopleAssignments>

            <htd:presentationElements>
              <htd:name xml:lang="en-US">
                Claim approval overdue
              </htd:name>
              <htd:name xml:lang="de-DE">
                Überfällige Schadensforderungsgenehmigung
              </htd:name>
            </htd:presentationElements>

          </htd:notification>

        </htd:escalation>
      </htd:completionDeadline>

      <htd:completionDeadline name="notifyDirector">
        <htd:documentation xml:lang="en-US">
          When not completed within 2 days after having been
          claimed, the functional director of claims processing is
          notified.
        </htd:documentation>
        <htd:for>P2D</htd:for>

        <htd:escalation name="severelyDelayedApproval">

          <htd:notification name="ClaimApprovalOverdue">
            <htd:documentation xml:lang="en-US">
              An inline defined notification using the approval data
              as its input.
            </htd:documentation>

            <htd:interface portType="cl:ClaimsHandlingPT"
              operation="escalate" />

            <htd:peopleAssignments>
              <htd:recipients>
                <htd:from logicalPeopleGroup="directorClaims">
                  <htd:argument name="clerkUserID">
                    htd:getActualOwner("ApproveClaim")
                  </htd:argument>
                </htd:from>
              </htd:recipients>
            </htd:peopleAssignments>

            <htd:presentationElements>
              <htd:name xml:lang="en-US">
                Claim approval severely overdue
              </htd:name>
              <htd:name xml:lang="de-DE">
                Hochgradig überfällige Schadensforderungsgenehmigung
              </htd:name>
            </htd:presentationElements>

          </htd:notification>

        </htd:escalation>
      </htd:completionDeadline>

    </htd:deadlines>

  </htd:task>

</htd:tasks>
```

```
  <htd:notifications>

    <htd:notification name="ClaimApprovalReminder">
      <htd:documentation xml:lang="en-US">
        This notification is used to remind people of pending
        out-dated claim approvals. Recipients of this notification
        maybe overriden when it is referenced.
      </htd:documentation>

      <htd:interface portType="cl:ClaimApprovalReminderPT"
        operation="notify" />

      <htd:peopleAssignments>
        <htd:recipients>
          <htd:from>
            <htd:literal>
              <htt:organizationalEntity>
                <htt:user>Alan</htt:user>
                <htt:user>Dieter</htt:user>
                <htt:user>Frank</htt:user>
                <htt:user>Gerhard</htt:user>
                <htt:user>Ivana</htt:user>
                <htt:user>Karsten</htt:user>
                <htt:user>Matthias</htt:user>
                <htt:user>Patrick</htt:user>
              </htt:organizationalEntity>
            </htd:literal>
          </htd:from>
        </htd:recipients>
      </htd:peopleAssignments>

      <htd:presentationElements>

        <htd:name xml:lang="en-US">Approve Claim</htd:name>
        <htd:name xml:lang="de-DE">
          Genehmigung der Schadensforderung
        </htd:name>

        <htd:presentationParameters>
          <htd:presentationParameter name="firstname"
            type="xsd:string">
            htd:getInput("firstname")
          </htd:presentationParameter>
          <htd:presentationParameter name="lastname"
            type="xsd:string">
            htd:getInput("lastname")
          </htd:presentationParameter>
          <htd:presentationParameter name="id" type="xsd:string">
            htd:getInput("taskId")
          </htd:presentationParameter>
        </htd:presentationParameters>

        <htd:subject xml:lang="en-US">
          Claim approval for $firstname$, $lastname$ is overdue. See
          task $id$.
        </htd:subject>

      </htd:presentationElements>

    </htd:notification>

  </htd:notifications>

</htd:humanInteractions>
```

J. Acknowledgements

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

Members of the BPEL4People Technical Committee:

Phillip Allen, Microsoft Corporation
Ashish Agrawal, Adobe Systems
Mike Amend, BEA Systems, Inc.
Stefan Baeuerle, SAP AG
Charlton Barreto, Adobe Systems
Justin Brunt, TIBCO Software Inc.
Martin Chapman, Oracle Corporation
Luc Clément, Active Endpoints, Inc.
Manoj Das, Oracle Corporation
Alireza Farhoush, TIBCO Software Inc.
Mark Ford, Active Endpoints, Inc.
Sabine Holz, SAP AG
Dave Ings, IBM
Gershon Janssen, Individual
Diane Jordan, IBM
Anish Karmarkar, Oracle Corporation
Ulrich Keil, SAP AG
Oliver Kieselbach, SAP AG
Matthias Kloppmann, IBM
Dieter König, IBM
Marita Kruempelmann, SAP AG
Frank Leymann, IBM
Mark Little, Red Hat
Alexander Malek, Microsoft Corporation
Ashok Malhotra, Oracle Corporation
Mike Marin, IBM
Vinkesh Mehta, Deloitte Consulting LLP
Jeff Mischkinsky, Oracle Corporation
Ralf Mueller, Oracle Corporation
Krasimir Nedkov, SAP AG
Benjamin Notheis, SAP AG
Michael Pellegrini, Active Endpoints, Inc.
Hannah Petereit, SAP AG
Gerhard Pfau, IBM
Karsten Ploesser, SAP AG

Ravi Rangaswamy, Oracle Corporation
Alan Rickayzen, SAP AG
Michael Rowley, BEA Systems, Inc.
Ron Ten-Hove, Sun Microsystems
Ivana Trickovic, SAP AG
Alessandro Triglia, OSS Nokalva
Claus von Riegen, SAP AG
Peter Walker, Sun Microsystems
Franz Weber, SAP AG
Prasad Yendluri, Software AG, Inc.

WS-HumanTask 1.0 Specification Contributors:

Ashish Agrawal, Adobe
Mike Amend, BEA
Manoj Das, Oracle
Mark Ford, Active Endpoints
Chris Keller, Active Endpoints
Matthias Kloppmann, IBM
Dieter König, IBM
Frank Leymann, IBM
Ralf Müller, Oracle
Gerhard Pfau, IBM
Karsten Plösser, SAP
Ravi Rangaswamy, Oracle
Alan Rickayzen, SAP
Michael Rowley, BEA
Patrick Schmidt, SAP
Ivana Trickovic, SAP
Alex Yiu, Oracle
Matthias Zeller, Adobe

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

K. Non-Normative Text

L. Revision History

[optional; should not be included in OASIS Standards]

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

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

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

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