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

Working Draft 02

~~13 March~~828 Julyne 2008

Specification URIs:

This Version:

http://docs.oasis-open.org/bpel4people/ws-humantask-1.1-spec-wd-02.html
http://docs.oasis-open.org/bpel4people/ws-humantask-1.1-spec-wd-02.doc
http://docs.oasis-open.org/bpel4people/ws-humantask-1.1-spec-wd-02.pdf

Previous Version:

http://docs.oasis-open.org/bpel4people/ws-humantask-1.1-spec-wd-01.html
http://docs.oasis-open.org/bpel4people/ws-humantask-1.1-spec-wd-01.doc
http://docs.oasis-open.org/bpel4people/ws-humantask-1.1-spec-wd-01.pdf

Latest Version:

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

Latest Approved Version:

N/A

Technical Committee:

OASIS BPEL4People TC

Chair:

Dave Ings, IBM

Editor(s):

Charlton Barreto, Adobe Systems
~~Mark Ford~~Luc Clément, Active Endpoints, Inc.
Dieter König, IBM
Vinkesh Mehta, Deloitte Consulting LLP
Ralf Mueller, Oracle Corporation
Krasimir Nedkov, SAP AG

Formatted: English (United Kingdom)

Formatted: Italian (Italy)

Ravi Rangaswamy, Oracle Corporation
Ivana Trickovic, SAP
Alessandro Triglia, OSS Nokalva

Formatted: Portuguese (Brazil)

Related work:

This specification is related to:

- WS-BPEL Extension for People (BPEL4People) Specification – Version 1.1

Declared XML Namespace(s):

WS-HumanTask namespaces (defined in this specification):

- **htd** – http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803
- ~~**htdp** – http://docs.oasis-open.org/ns/bpel4people/ws-humantask/protocol/200803~~
- **~~htd~~hta** – ~~http://docs.oasis-open.org/ns/bpel4people/ws-humantask/api/services/200803~~http://docs.oasis-open.org/ns/bpel4people/ws-humantask/api/200803
- **~~htd~~htt** – ~~http://docs.oasis-open.org/ns/bpel4people/ws-humantask/api/types/200803~~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

Field Code Changed

Formatted: English (United Kingdom)

Formatted: Bullets and Numbering

Other 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

Abstract:

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

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

Status:

This document was last revised or approved by the [TC name | membership of OASIS] on the above date. The level of approval is also listed above. Check the “Latest Version” or “Latest Approved Version” location noted above for possible later revisions of this document.

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

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

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

Notices

Table of Contents

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 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 on timeouts, triggering an appropriate escalation action.

This also holds true for *notifications*. Notifications are 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 that 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.

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 HT, 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

2.2 Notational Conventions

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].

2.3 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 must understand the extension. If an optional extension is used, a compliant implementation may ignore the extension.

2.4 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.

2.4.1 Syntax

```
<htd:humanInteractions
  xmlns:htd="~~http://www.example.org/WS-HT~~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>
```

2.4.2 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 WS-HumanTask. The WS-HumanTask constructs that use expressions may override the default expression language for individual expressions. A WS-HumanTask compliant implementation 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 WS-HumanTask. The WS-HumanTask constructs that use query expressions may override the default query language for individual query expressions. A WS-HumanTask compliant implementation 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 implementation does not support one or more of the extensions with mustUnderstand="yes", then the human interactions definition MUST be rejected. Optional extensions MAY be ignored. It is not required to declare optional extension. 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. 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. Any number of `<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 process but makes no statement about where those definitions may be found. The `location` attribute is a hint and a WS-HumanTask compliant implementation 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://www.example.org/WS-HT`~~`http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803` when importing WS-HumanTask documents, or to `http://schemas.xmlsoap.org/wsdl/` when importing WSDL 1.1 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 may be found.

A human interactions definition MUST import all WS-HumanTask and WSDL definitions it uses. In order to support the use of definitions from namespaces spanning multiple documents, a human interactions 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 human interactions definition MUST be rejected if the imported documents contain conflicting definitions of a component used by the importing process definition.

Documents (or namespaces) imported by an imported document (or namespace) MUST NOT be transitively imported by a WS-HumanTask compliant implementation. In particular, this means that if an external item is used by a task enclosed in the human interactions definition, then a document (or namespace) that defines that item MUST be directly imported by the human interactions 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 ~~all~~ logical people groups ~~used in the enclosing human tasks and notifications~~. 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~~The reference attribute specifies logical people group, in~~ case a logical people group ~~is~~ used in the humanInteractions element ~~that~~ is defined in an ~~elsewhere~~imported WS-HumanTask definition, the reference attribute MUST be used to specify the logical people group. ~~The reference attribute is optional.~~ 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 5 “Notifications”.

Element humanInteractions MUST NOT be empty, that is it MUST include at least one element.

All WS-HumanTask elements 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>
```

Formatted: French (Canada)

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. Depending on how the task has been instantiated the task initiator may or may not 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. Compliant implementations 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.

Excluded owners may not become an actual or potential owner and thus they may not reserve or start the task.

An *actual owner* of a task is the person actually performing the task. A task has exactly one actual owner. 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.

Business administrators play the same role as task stakeholders but at task type level. Therefore, business administrators can perform the exact same operations as task stakeholders. Business administrators may also observe the progress of notifications. Compliant implementations 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.

3.2 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 3.2.1 “Using Logical People Groups”)
- Via literals (see 3.2.2 “Using Literals”)
- Via expressions e.g., by retrieving data from the input message of the human task (see 3.2.3 “Using Expressions”).

When specifying people assignments then the data type `htd:tOrganizationalEntity` is used. Using `htd:tOrganizationalEntity` allows to assign either a set of people or an unresolved group of people (“work queue”).

Syntax:

```
<htd:peopleAssignments>

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

</htd:peopleAssignments>
```

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

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

<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>
```

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

3.2.1 Using Logical People Groups

A *logical people group* represents either 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 support query parameters which are passed to the people query at runtime. Parameters may refer to task instance data (see section 3.4 for more details). During people query execution an infrastructure may decide which of the parameters defined by the logical people group are used. It 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.

People queries are evaluated during the creation of a human task or a notification. If a people query fails then the human task or notification is created anyway. Failed people queries are treated like people queries that return an empty result set. If the potential owner people query returns an empty set of people then nomination has to be performed (see section 4.7.1 “Normal processing of a Human Task”). In case of notifications, the same applies to notification recipients.

People queries return either one person, a set of people, or the name of one or many groups of people. The latter is added to support “work queue” based business scenarios, where people see work they have been assigned to due to their membership of a certain group. Especially in cases where group membership changes frequently, this “late binding” to the actual group members is beneficial.

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 may 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 is used.

Example:

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

3.2.2 Using Literals

People assignments can be defined literally by directly specifying the user identifier(s) or the name(s) of groups using either the `htd:tOrganizationalEntity` or `htd:tUser` data type introduced below (see 3.2.4 “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>
      <htd:organizationalEntity>
        <htd:users>
          <htd:user>Alan</htd:user>
          <htd:user>Dieter</htd:user>
          <htd:user>Frank</htd:user>
          <htd:user>Gerhard</htd:user>
          <htd:user>Ivana</htd:user>
          <htd:user>Karsten</htd:user>
          <htd:user>Matthias</htd:user>
          <htd:user>Patrick</htd:user>
        </htd:users>
      </htd:organizationalEntity>
    </htd:literal>
  </htd:from>
</htd:potentialOwners>
```

Example specifying group names:

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

3.2.3 Using Expressions

Alternatively people can be assigned using expressions returning either an instance of the `htd:tOrganizationalEntity` data type or the `htd:tUser` data type introduced below (see 3.2.4 “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 is used.

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>
```

3.2.4 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 may 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>
    <xsd:element ref="users" />
    <xsd:element ref="groups" />
  </xsd:choice>
</xsd:complexType>

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

<xsd:element name="users" type="tUserlist" />
<xsd:complexType name="tUserlist">
  <xsd:sequence>
    <xsd:element ref="user" minOccurs="0" maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>

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

<xsd:element name="groups" type="tGrouplist" />
<xsd:complexType name="tGrouplist">
  <xsd:sequence>
```

```
    <xsd:element ref="group" minOccurs="0" maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>
```

3.3 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 engine 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 may 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 may comprise and manipulate documents other than the task instance. It can be specified by a rendering method within the task description.

3.4 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.

3.4.1 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.

3.4.2 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

An implementation may extend this set of properties available in the task context. For example, the actual owner may start the execution of the task but the task could be long-running task so intermediate state could be saved in the task context.

3.4.3 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.

3.4.3.1 Ad-hoc Attachments

Arbitrary additional data may 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.

The `name` element is used to specify attachment name. Several attachments may have the same name and can then be retrieved as a collection.

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 type.

The `accessType` element indicates if the attachment is specified inline or by reference. In the inline case it contains 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 contains 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 `attachedAt` 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.

A task may have ad-hoc attachments. Ad-hoc attachments can be added, deleted and retrieved by name. Deletion and retrieving affects all attachments of that name.

Attachment Info Data Type

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

```
<xsd:element name="attachmentInfo" type="tAttachmentInfo" />
<xsd:complexType name="tAttachmentInfo">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string" />
    <xsd:element name="accessType" type="xsd:string" />
    <xsd:element name="contentType" type="xsd:string" />
    <xsd:element name="attachedAt" type="xsd:dateTime" />
    <xsd:element name="attachedBy" type="htd: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>
```

3.4.3.2 Comments

A task may 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 the text, user information and a timestamp. Comments are usually added individually, but retrieved as one group. Comments usage is optional in a task.

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

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

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="addedAt" type="xsd:dateTime" />
    <xsd:element name="addedBy" 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.

3.4.4 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 ~~`htd:taskAbstract`~~`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 ~~`htd:task`~~`htt:tTask` contains the data of a task or a notification, except ad-hoc attachments, comments and presentation description. The data that is not contained in ~~`htd:task`~~`htt:tTask` may be retrieved separately from the task engine 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 (`startBy` and `complete By`) of the individual deadlines can be retrieved using the query operation (see section 6.1.3 “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="xsd:nonNegativeInteger" minOccurs="0" />
    <xsd:element name="createdOn"
                 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="startByExists"
                 type="xsd:boolean" minOccurs="0" />
    <xsd:element name="completeByExists"
                 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:any namespace="##other" processContents="lax"
             minOccurs="0" maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>
```

Task Data Type

```
<xsd:element name="task" type="tTask"/>
<xsd:complexType name="tTask">
  <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="xsd:nonNegativeInteger" minOccurs="0"/>
    <xsd:element name="taskInitiator"
                 type="htd:tUser" minOccurs="0"/>
    <xsd:element name="taskStakeholders"
                 type="htd:tOrganizationalEntity" minOccurs="0"/>
    <xsd:element name="potentialOwners"
                 type="htd:tOrganizationalEntity" minOccurs="0"/>
    <xsd:element name="businessAdministrators"
                 type="htd:tOrganizationalEntity" minOccurs="0"/>
    <xsd:element name="actualOwner"
                 type="htd:tUser" minOccurs="0"/>
    <xsd:element name="notificationRecipients"
                 type="htd:tOrganizationalEntity" minOccurs="0"/>
    <xsd:element name="createdOn"
                 type="xsd:dateTime"/>
    <xsd:element name="createdBy"
                 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="startByExists"
                 type="xsd:boolean" minOccurs="0"/>
    <xsd:element name="completeByExists"
                 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="primarySearchBy"
                 type="xsd:string" 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: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>
```

4 Human Tasks

The `<task>` element is used to specify human tasks. The section below 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">

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

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

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

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

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

  <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>*
      ...
    </htd:startDeadline>

    <htd:completionDeadline>*
      ...
    </htd:completionDeadline>

  </htd:deadlines>
```

```
</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 of a task element is used to uniquely identify the task definition. This attribute is mandatory. It is not used for task rendering.
- `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 are mandatory. 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 callback one-way operation MUST be specified, 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 8 "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 not present, the priority of the task is unspecified. 0 is the highest priority, larger numbers identify lower priorities. The result of the expression evaluation is of type `xsd:integer`. 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 mandatory. See section 0 for more details on people assignments.
- `delegation`: This element is used to specify constraints concerning delegation of the task. Attribute `potentialDelegatees` defines to whom the task may be delegated. The following values are allowed:
    - `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 may 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 mandatory.
- `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 4.6 for more details on timeouts and escalations.

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 should 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 may 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. It is not allowed to 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 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. Parameters MUST be 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 and not arbitrary expressions are allowed to be embedded with 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. It is not allowed to 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 compliant implementation MUST support the content type “text/plain”. It SHOULD support HTML (such as “text/html” or “application/xml+xhtml”). There could be zero or more `description` elements. As descriptions may exist with different content types, it is allowed to specify multiple `description` elements having the same value for attribute `xml:lang`, but their content types MUST be different.

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>
```

Formatted: French (France)

Formatted: French (France)

Formatted: German (Germany)

4.4 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 may be provided in a task definition or a notification definition. A task or notification can be deployed on any compliant implementation, 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 may 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 6.2 may or may not be evaluated by a compliant implementation.

Syntax:

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

4.5 Elements for People Assignment

The `<peopleAssignments>` element is used to assign people to the task. For each generic human role, a people assignment element can be specified. For human tasks it is mandatory to specify people assignment for potential owners. If no potential owner should be assigned by the human task's definition, e.g. because nomination is used, then this is accomplished by adding an empty `<potentialOwners>` element. Specifying people assignments for task stakeholders, task initiators, excluded owners and business administrators is optional. Human tasks never specify recipients. People assignments for actual owners MUST NOT be specified.

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 may result in a set of values or an empty set. In case people assignment results in an empty set then the task may require administrative attention. This is out of scope of the specification, except for people assignments for potential owners (see section 4.7.1 “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>
```

4.6 Elements for Handling Timeouts and Escalations

Timeouts and escalations allow the specification of a date or time before which the task must 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 must be reached with the given duration or by the given date. They are:

- Start deadline: Specifies the time until the task must start, i.e. it is must reach state *InProgress*. It is defined as either the period of time or the point in time until the task must 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, at least one deadline MUST be defined.

Syntax:

```
<htd:deadlines>

  <htd:startDeadline>*

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

Formatted: French (France)

```
    ( <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>*
    ...
  </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 5 “Notifications”. An escalation MUST specify exactly one escalation action.

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

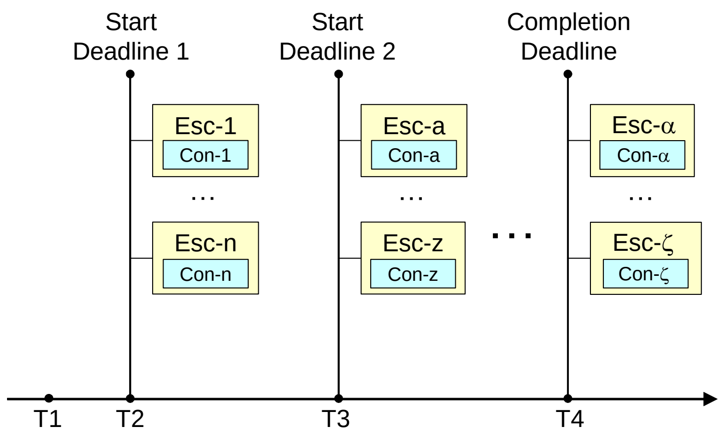
- A notification defined in the `<humanInteractions>` root element or imported from a different namespace can be referenced by specifying its QName in the `reference` attribute of a `<localNotification>` element. When referring to a notification, the priority and the people assignments of the original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

Notifications used in escalations may use the same type of input data as the surrounding task, or different type of data. If the same type of data is used then the input message of the 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.

There MUST be 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, they MUST be executed 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, the element MUST specify potential owners.

In the case where several reassignment escalations are triggered, the first reassignment (lexical order) will be considered for execution. 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 may involve an overlapping set of people. The completion deadline would be missed at time T4.

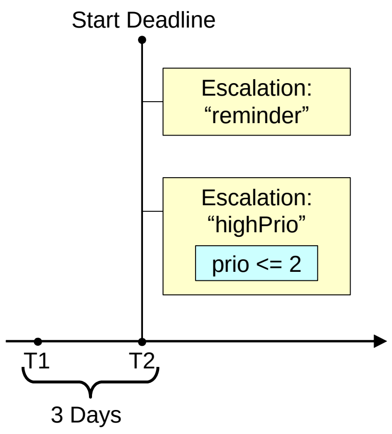
Syntax:

```
<htd:deadlines>

  <htd:startDeadline>*
    ...

    <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>*
    ...
  </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>
  <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:toPart name="taskId">~~
~~        htd:getTaskID("ApproveClaim")~~
```

```
      ~~</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>

```

Formatted: German (Germany)

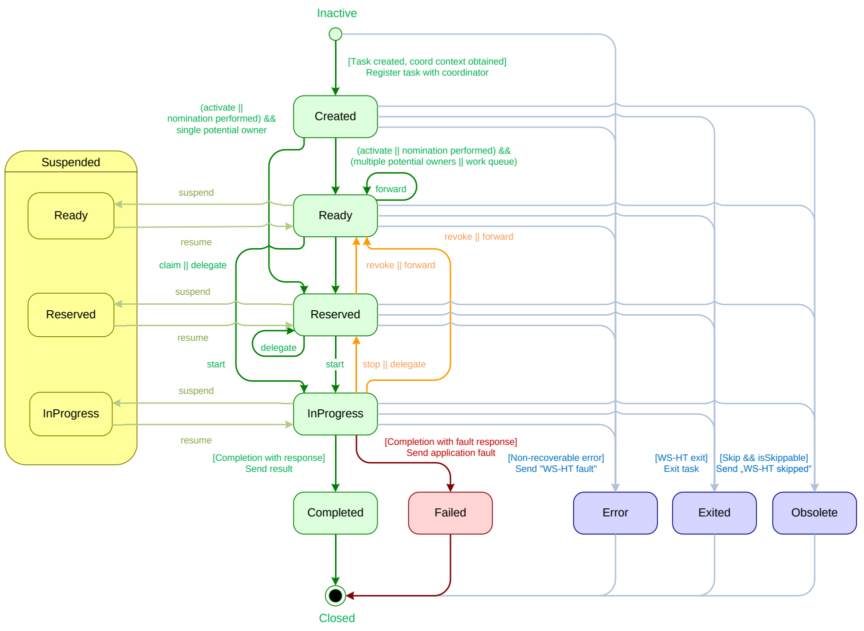
Formatted: German (Germany)

```
    </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>
            <htd:organizationalEntity>
              <htd:users>
                <htd:user>Alan</htd:user>
              </htd:users>
            </htd:organizationalEntity>
          </htd:literal>
        </htd:from>
      </htd:potentialOwners>

    </htd:reassignment>

  </htd:escalation>

</htd:startDeadline>
```

Formatted: German (Germany)

4.7 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.


Inactive
[Task created, coord context obtained] Register task with coordinator
Created
(activate || nomination performed) && single potential owner
(activate || nomination performed) && (multiple potential owners || work queue)
Suspended
Ready
Reserved
InProgress
suspend
resume
forward
revoke || forward
claim || delegate
delegate
start
stop || delegate
[Completion with response] Send result
[Completion with fault response] Send application fault
[Non-recoverable error] Send "WS-HT fault"
[WS-HT exit] Exit task
[Skip && isSkippable] Send „WS-HT skipped"
Completed
Failed
Error
Exited
Obsolete
Closed

4.7.1 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, 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.

4.7.2 Releasing a Human Task

The current actual owner of a human task may *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.

4.7.3 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.

Similarily, 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.

4.7.4 Suspending and resuming a Human Task

In any of its active states (*Ready*, *Reserved*, *InProgress*), a task can be suspended, transitioning it into the *Suspended* state. The *Suspended* state has sub-states to indicate the original state of the task.

On resumption of the task, it transitions back to the original state from which it had been suspended.

4.7.5 Skipping a Human Task

A person working on a human task or a business administrator may decide that a task is no longer needed, and hence skip this task. This transitions the task into the *Obsolete* state. This is considered a "good" outcome of a task, even though an empty result is returned. The enclosing environment can be notified of that transition as described in section 0.

The task can only be skipped if this capability is specified during the task invocation. A side-effect of this is that a task which is invoked using basic Web service protocols is not skipable.

4.7.6 Termination of a Human Task

The enclosing environment of a human task (such as the calling application or business process) may decide that a task is no longer needed and terminate it, either because a timeout has reached in that enclosing context (i.e., the task has expired), or because the enclosing environment itself is terminated. These events transition the task into the *Obsolete* state.

4.7.7 Error handling for Human Task

If a human task encounters a non-recoverable error in any of its state (for example, it executes a divide by zero in an XPath expression), it transitions into the *Error* state. This is considered a “bad” outcome of the task and no result is returned. The enclosing environment can be notified of that transition as described in section 0.

5 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 may 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.

5.1 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>
```

5.2 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 of a notification element is used to uniquely identify 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. The operation MUST be 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 not present, the priority of the task is unspecified. 0 is the highest priority, larger numbers identify lower priorities. The result of the expression evaluation is of type `xsd:integer`. 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. The element 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.

5.3 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*.

6 Programming Interfaces

6.1 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 compliant implementation MUST provide the operations listed below and an application (such as a task list client) may use these operations to manipulate the task. All operations are executed in a synchronous fashion and return faults provided that certain preconditions do not hold. The response message resulting from an operation invocation may be void. 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 results in the ~~illegal~~hta:illegalArgumentFault being thrown. Invoking an operation that is not allowed in the current state of the task results in an ~~illegal~~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 ~~illegal~~hta:illegalAccessFault and ~~recipientNotAllowed~~hta:recipientNotAllowed is thrown 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) results in an ~~illegal~~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 4.7 “Human Task Behavior and State Transitions”. For a formal definition of the allowed operations, see WS-HumanTask Data Types Schema

Formatted: Normal

Note to specification editors: the WS-HumanTask data types XML Schema definition is separately maintained in artifact

ws-humantask-types.xsd

Formatted: Normal

The contents of this artifact shall be copied back into this section before publishing the specification, e.g., as a committee draft.

Formatted: Font: Arial

WS-HumanTask API ~~Operations WSDL~~.

Formatted: Font: Courier New

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).

6.1.1 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. The participant operations listed below only apply to tasks unless explicitly noted otherwise. The authorization column indicates people of which roles are authorized to perform the operation. Stakeholders of the task are not mentioned explicitly. They have the same authorization rights as business administrators.

Operation Name	Description	Parameters	Authorization
claim	Claim responsibility for a task, i.e. set the task to status *Reserved*	In • task identifier Out • void	Potential Owners Business Administrator
start	Start the execution of the task, i.e. set the task to status *InProgress*.	In • task identifier Out • void	Actual Owner Potential Owners (state *Ready*)
stop	Cancel/stop the processing of the task. The task returns to the *Reserved* state.	In • task identifier Out • void	Actual Owner Business Administrator
release	Release the task, i.e. set the task back to status *Ready*.	In • task identifier Out • void	Actual Owner Business Administrator
suspend	Suspend the task.	In • task identifier Out • void	Potential Owners (state *Ready*) Actual Owner Business Administrator
suspendUntil	Suspend the task for a given period of time or until a fixed point in time. The caller has to specify either a period of time or a fixed point in time.	In • task identifier • time period • point of time Out • void	Potential Owners (state *Ready*) Actual Owner Business Administrator
resume	Resume a suspended task.	In • task identifier	Potential Owners (state *Ready*) Actual Owner

		Out • void	Business Administrator
complete	Execution of the task finished successfully. If no output data is set the operation returns ~~illegal~~hta:illegalArgumentFault.	In • task identifier • output data of task Out • void	Actual Owner
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	Notification Recipient
fail	Actual owner completes the execution of the task raising a fault. The fault ~~illegal~~hta:illegalOperationFault is returned if the task interface defines no faults. If fault name or fault data is not set the operation returns ~~illegal~~hta:illegalArgumentFault.	In • task identifier • fault name • fault data Out • void	Actual Owner
setPriority	Change the priority of the task. The caller has to specify the integer value of the new priority.	In • task identifier • priority Out • void	Actual Owner Business Administrator
addAttachment	Add attachment to a task.	In • task identifier • attachment name • access type • attachment Out • void	Actual Owner Business Administrator

getAttachmentInfos	Get attachment information for all attachments associated with the task.	In • task identifier Out • list of attachment data (list of htt~~htd~~:attachmentInfo)	Potential Owners Actual Owner Business Administrator
getAttachments	Get all attachments of a task with a given name.	In • task identifier • attachment name Out • list of attachments (list of ~~htd~~htt:attachment)	Potential Owners Actual Owner Business Administrator
deleteAttachments	Delete the attachments with the specified name from the task (if multiple attachments with that name exist, all are deleted). Attachments provided by the enclosing context are not affected by this operation.	In • task identifier • attachment name Out • void	Actual Owner Business Administrator
addComment	Add a comment to a task.	In • task identifier • plain text Out • void	Potential Owners Actual Owner Business Administrator
getComments	Get all comments of a task	In • task identifier Out • list of comments (list of ~~htd~~htt:comment)	Potential Owners Actual Owner Business Administrator
skip	Skip the task. If the task is not skipable then the fault ~~illegal~~hta:illegalOperationFault is returned.	In • task identifier Out • void	Task Initiator Actual Owner Business Administrator
forward	Forward the task to	In	Potential Owners

	another organization entity. The caller has to specify the receiving organizational entity. Potential owners can only forward a task while the task is in the *Ready* state. For details on forwarding human tasks refer to section 4.7.3.	• task identifier • organizational entity (`htd:tOrganizationalEntity`) Out • void	Actual Owner Business Administrator
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 is added to the set of potential owners. For details on delegating human tasks refer to section 4.7.3.	In • task identifier • organizational entity (`htd:tOrganizationalEntity`) Out • void	Potential Owners (only in *Ready* state) Actual Owner Business Administrator
getRendering	Applies to both tasks and notifications. Returns the rendering specified by the type parameter.	In • task identifier • rendering type Out • any type	Any
getRenderingTypes	Applies to both tasks and notifications. Returns the rendering types available for the task or notification.	In • task identifier Out • list of QNames	Any
getTaskInfo	Applies to both tasks and notifications. Returns a data object of type `htt:tTask`	In • task identifier Out • task (~~htd~~`htt:tTask`)	Any
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	Any
setOutput	Set the data for the part of the task's output message.	In • task identifier • part name (optional for	Actual Owner

		single part messages) • output data of task Out • void	
deleteOutput	Deletes the output data of the task.	In • task identifier Out • void	Actual Owner
setFault	Set the fault data of the task. The fault ~~illegal~~hta:illegalOperationFault is returned if the task interface defines no faults.	In • task identifier • fault name • fault data of task Out • void	Actual Owner
deleteFault	Deletes the fault name and fault data of the task.	In • task identifier Out • void	Actual Owner
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	Potential Owners Actual owner Business Administrator
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	Actual Owner Business Administrator
getFault	Get the fault data of the task.	In • task identifier Out • fault name • fault data	Actual Owner Business Administrator
getOutcome	Get the outcome of the task	In • task identifier Out	Any

Formatted: Bullets and Numbering

		• string	

Formatted: Bullets and Numbering

All these operations MUST be supported by a compliant implementation.

6.1.2 Simple Query Operations

Simple query operations allow to retrieve task data. These operations MUST be supported by a compliant implementation. The identity of the user is implicitly passed when invoking any of the following operations.

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 work queue has been specified then only personal tasks are returned. If the work queue is specified then only tasks of that work queue are returned. The *where* clause may only 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 *where* clause is logically ANDed with the created-on clause, which may only reference the column Task.CreatedOn 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 will not exceed this limit.	In • task type ("ALL" | "TASKS" | "NOTIFICATIONS") • generic human role • work queue • status list • where clause • created-on clause • maxTasks Out • list of tasks (list of ~~`htd`~~`htt:tTaskAbstract`)	Any

getMyTasks	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 work queue has been specified then only personal tasks are returned. If the work queue is specified then only tasks of that work queue are returned. The *where* clause may only 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 *where* clause is logically ANDed with the created-on clause, which may only reference the column Task.CreatedOn with operators as described above. The combination of the two clauses enables simple but restricted paging inthe task list client. If maxTasks is specified, then the number of task details returned for this query will not exceed this limit.	In • task type ("ALL" | "TASKS" | "NOTIFICATIONS") • generic human role • work queue • status list • where clause • created-on clause • maxTasks Out • list of tasks (list of ~~htd~~htt:tTask)	Any

The return types tTaskAbstract and tTask are defined in section 3.4.4 "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 4.7 “Human Task Behavior and State Transitions”)
Priority	xsd:nonNegativeInteger (0 = highest)
CreatedOn	xsd:dateTime
ActivationTime	xsd:dateTime
ExpirationTime	xsd:dateTime
HasPotentialOwners	xsd:boolean
StartByExists	xsd:boolean
CompleteByExists	xsd:boolean
RenderMethExists	xsd:boolean
Escalated	xsd:boolean
PrimarySearchBy	xsd:string

6.1.3 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 6.1.2. A compliant implementation MAY support this operation. An implementation MAY restrict the results according to authorization of the invoking user.

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	In • select clause • where clause • order-by clause • maxTasks

	will 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.	• taskIndexOffset Out • query result (~~htd:taskQueryResultSet~~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="xsd:nonNegativeInteger"/>
    <xsd:element name="taskInitiator"
                 type="htd:tUser"/>
    <xsd:element name="taskStakeholders"
                 type="htd:tOrganizationalEntity"/>
    <xsd:element name="potentialOwners"
                 type="htd:tOrganizationalEntity"/>
    <xsd:element name="businessAdministrators"
                 type="htd:tOrganizationalEntity"/>
    <xsd:element name="actualOwner" type="htd:tUser"/>
    <xsd:element name="notificationRecipients"
                 type="htd:tOrganizationalEntity"/>
    <xsd:element name="createdOn" type="xsd:dateTime"/>
    <xsd:element name="createdBy" 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="startByExists" type="xsd:boolean"/>
    <xsd:element name="completeByExists" type="xsd:boolean"/>
    <xsd:element name="presentationName" type="tPresentationName"/>
    <xsd:element name="presentationSubject"
```

```
                 type="tPresentationSubject"/>
    <xsd:element name="renderingMethodExists" type="xsd:boolean"/>
    <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="primarySearchBy" type="xsd:string"/>
    <xsd:element name="outcome" type="xsd:string"/>
    <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 compliant implementation 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 4.7 "Human Task Behavior and State Transitions"
Priority	xsd:int (0 = highest)	
UserId	xsd:string	
Group	xsd:string	
GenericHumanRole	xsd:string	
CreatedOn	xsd:dateTime	The time in UTC when the task has been created.

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.
Skipable	xsd:boolean	
StartBy	xsd:dateTime	The time in UTC when the task should have been started. This time corresponds to the respective start deadline.
CompleteBy	xsd:dateTime	The time in UTC when the task should have been completed. This time corresponds to the respective end deadline.
PresentationName	xsd:string	The task's presentation name.
PresentationSubject	xsd:string	The task's presentation subject.
RenderingMethodName	xsd:Qname	The task's rendering method name.
FaultMessage	xsd:any	
InputMessage	xsd:any	
OutputMessage	xsd:any	
AttachmentName	xsd:string	
AttachmentType	xsd:string	
Escalated	xsd:boolean	
PrimarySearchBy	xsd:string	
Outcome	xsd:string	

6.1.4 Administrative Operations

Operations to be executed for administrative purposes. Actual definition of authorization for operations is outside the scope of this specification.

Operation Name	**Description**	**Parameters**	**Authorization**
activate	Activate the task, i.e. set the task to status *Ready.*	In • task identifier Out	Business Administrator

		• void	
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 *Ready*. This can only be performed when the task is in the state *Created*.	In • task identifier • organizational entity (`htd:`~~`tOrganizationalType`~~`tOrganizationalEntity`) Out • void	Business Administrator
setGenericHumanRole	Replace the organizational assignment to the task in one generic human role.	In • task identifier • generic human role • organizational entity (`htd:`~~`tOrganizationalType`~~`tOrganizationalEntity`) Out • void	Business Administrator

6.2 XPath Extension Functions

This section introduces~~The following~~ XPath extension functions that are provided to be used within the definition of a human task or notification. When defining properties using these XPath functions note the initialization order in section 4.7.1.

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. Evaluates to an empty `htd:organizationalEntity` in case of an error.	In • task name (optional) Out • potential owners

	If the task name is not present the current task is considered.	(`htd:organizationalEntity`)
getActualOwner	Returns the actual owner of the task. Evaluates to an empty `htd:user` in case there is no actual owner. If the task name is not present the current task is considered.	In • task name (optional) Out • the actual owner (user id as `htd:user`)
getTaskInitiator	Returns the initiator of the task. Evaluates to an empty `htd:user` in case there is no initiator. If the task name is not present the current task is considered.	In • task name (optional) Out • the task initiator (user id as `htd:user`)
getTaskStakeholders	Returns the stakeholders of the task. Evaluates to an empty `htd:organizationalEntity` in case of an error. If the task name is not present the current task is considered.	In • task name (optional) Out • task stakeholders (`htd:organizationalEntity`)
getBusinessAdministrators	Returns the business administrators of the task. Evaluates to an empty `htd:organizationalEntity` in case of an error. If the task name is not present the current task is considered.	In • task name (optional) Out • business administrators (`htd:organizationalEntity`)
getExcludedOwners	Returns the excluded owners. Evaluates to an empty `htd:organizationalEntity` in case of an error. If the task name is not present the current task is considered.	In • task name (optional) Out • excluded owners (`htd:organizationalEntity`)
getTaskPriority	Returns the priority of the task. Evaluates to "-1" in case of an error. If the task name is not present the current task is considered.	In • task name (optional) Out • priority (xsd:nonNegativeInteger)

getInput	Returns the part of the task's input message. If the task name is not present the current task is considered.	In • part name • task name (optional) Out • input message
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 `htd:organizationalEntity` contains an empty user list. If the task name is not present the current task is considered.	In • task name (optional) • name of the logical people group Out • the value of the logical people group (`htd:organizationalEntity`)
getOutcome	Returns the outcome of the task. Evaluates to an empty string in case there is no outcome specified for the task. If the task name is not present the current task is 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 (`htd:organizationalEntity |htd:users |htd:user`) • set2 (`htd:organizationalEntity |htd:users |htd:user`) Out • result (htd:organizationalEntity)
intersect	Constructs an organizationalEntity containing every user that occurs in **both set1 and set2**, eliminating duplicate users.	In • set1 (`htd:organizationalEntity |htd:users |htd:user`) • set2 (`htd:organizationalEnt`

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Font: Courier New

		`ity` `|htd:users` `|htd:user`) Out • result (`htd: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 ("4-eyes principle").	In • set1 (`htd:organizationalEntity` `|htd:users` `|htd:user`) • set2 (`htd:organizationalEntity` `|htd:users` `|htd:user`) Out • result (`htd:organizationalEntity`)

1898

7 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 is an interoperable manner, a coordination protocol, namely the *WS-HT coordination protocol*, is introduced to exchange life-cycle command messages between an application and an invoked human task. A simplified protocol applies to notifications.

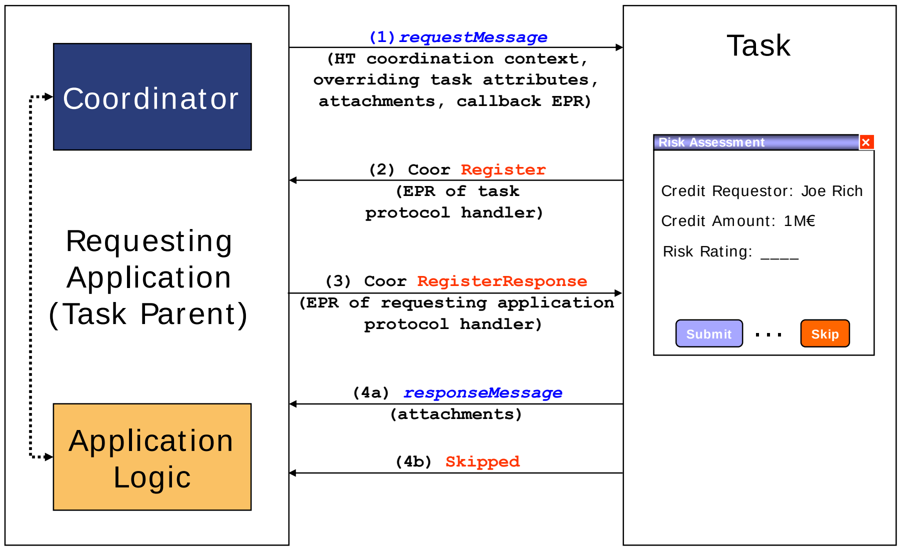

Coordinator
Requesting Application (Task Parent)
Application Logic
(1) requestMessage (HT coordination context, overriding task attributes, attachments, callback EPR)
(2) Coor Register (EPR of task protocol handler)
(3) Coor RegisterResponse (EPR of requesting application protocol handler)
(4a) responseMessage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

Figure 1: ***Message Exchange between Application and Human Task***

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-HT coordination protocol on the application's behalf.

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

<u>Scenario 1:</u> At some point in time, the application invokes the human task through its service interface. In order to signal to the human task infrastructure that an instance of the human task should 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-HT 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-HT coordination type ~~`http://www.example.org/WS-`~~

~~HT/protocol~~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 its hosting WS-HumanTask (WS-HT) implementation. The coordination context further contains in its `RegistrationService` element an endpoint reference that the WS-HT implementation hosting the human task must use to register the task as 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 calling application 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.

Formatted: Bullets and Numbering

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

Now the instance of the human task is activated, so the assigned person can perform the task (e.g. the risk assessment). Once the human task was successfully completed, a response message is passed back to the parent application (step (4a) in Figure 1).

Scenario 2: If the human task is not completed with a result, but the assigned person determines that the task should rather be skipped (and hence reaches its *Obsolete* final state), a “`skipped`” coordination protocol message is sent from the human task to its parent application (step (4b) in Figure 1). No response message is passed back.

Scenario 3: If the parent application needs to end prematurely before the invoked human task has been completed, it sends an `exit` coordination protocol message to the human task, causing the human task to end its processing. No response message is passed back.

In case of notifications, only some of the overriding attributes are propagated with the request message. Only priority and people assignments can be overridden for a notification, and the elements isSkipable, expirationTime and attachments are ignored if present. Likewise, the WS-HT coordination context, attachments and the callback EPR do not apply to notifications and are ignored as well. Finally, a notification does not return WS-HT coordination protocol messages. There is no message exchange beyond the initiating request message.

7.1 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. Human task aware requesting applications MUST support WS-HT protocol messages in addition to application requesting, responding and fault messages.

See diagram in section 4.7 “Human Task Behavior and State Transitions”.

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

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*

7.2 Protocol Messages

All WS-HT protocol messages have the following type:

```
<xsd:complexType name="ProtocolMsgType">
  <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.

7.2.1 Protocol Messages Received by a Task Parent

The following is the definition of the ~~htdp~~htcp:skipped message.

```
<xsd:element name="skipped"
type="~~spc:ProtocolMsgType~~htcp:ProtocolMsgType" />
<wsdl:message name="skipped">
  <wsdl:part name="parameters" element="~~htdp~~htcp:skipped" />
</wsdl:message>
```

The ~~htdp~~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 ~~htdp~~htcp:fault message.

```
<xsd:element name="fault" type="~~spc:ProtocolMsgType~~htcp:ProtocolMsgType"
/>
<wsdl:message name="fault">
  <wsdl:part name="parameters" element="~~htdp~~htcp:fault" />
</wsdl:message>
```

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

7.2.2 Protocol Messages Received by a Task

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

```
<xsd:element name="exit" type="~~spc:ProtocolMsgType~~htcp:ProtocolMsgType"
/>
<wsdl:message name="exit">
  <wsdl:part name="parameters" element="~~htdp~~htcp:exit" />
</wsdl:message>
```

7.3 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-HT coordination protocol.

7.3.1 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-HT 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 can be passed to the tasked parent by using the operation named `skippedOperation`.

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

7.3.2 Protocol Endpoint of the Task

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-HT 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 can be passed to the subprocess by using the operation named `exitOperation`.

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

7.4 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.

7.4.1 Schema of the Human Task Context

The following describes the XML schema representation of the task context:

```
<xsd:element name="humanTaskContext" type="tHumanTaskContext" />
<xsd:complexType name="tHumanTaskContext">
  <xsd:sequence>
    <xsd:element name="priority" type="xsd:nonNegativeInteger"
      minOccurs="0" />
    <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="attachments" type="tAttachments" minOccurs="0" />
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="tPeopleAssignments">
  <xsd:sequence>
    <xsd:group ref="genericHumanRole" minOccurs="0"
      maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>

<xsd:group name="genericHumanRole">
  <xsd:choice>
    <xsd:element ref="potentialOwners" />
    <xsd:element ref="excludedOwners" />
    <xsd:element ref="taskInitiator" />
    <xsd:element ref="taskStakeholders" />
    <xsd:element ref="businessAdministrators" />
    <xsd:element ref="recipients" />
  </xsd:choice>
</xsd:group>
<xsd:element name="potentialOwners" type="tGenericHumanRole" />
<xsd:element name="excludedOwners" type="tGenericHumanRole" />
<xsd:element name="taskInitiator" type="tGenericHumanRole" />
<xsd:element name="taskStakeholders" type="tGenericHumanRole" />
<xsd:element name="businessAdministrators" type="tGenericHumanRole" />
<xsd:element name="recipients" type="tGenericHumanRole" />
<xsd:complexType name="tGenericHumanRole">
  <xsd:sequence>
    <xsd:element ref="htd:organizationalEntity" />
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="tAttachments">
  <xsd:sequence>
    <xsd:element name="returnAttachments" type="tReturnAttachments"
      minOccurs="0" />
    <xsd:element ref="~~htda~~hta: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>
```

Formatted: English (United States)

7.4.2 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 `humanTaskContext` element is simply mapped to a single SOAP header as a whole. The following listing shows the SOAP binding of the human task context in an infoset representation.

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
            xmlns:~~htdp~~htc="~~http://www.osoa.org/WS-
HT/protocol~~http://docs.oasis-open.org/ns/bpel4people/ws-
humantask/context/200803">
 <S:Header>
   <~~htdp~~htc:humanTaskContext>
     <~~htdp~~htc:priority>...</~~htdp~~htc:priority>?
     <~~htdp~~htc:peopleAssignments>...</~~htdp~~htc:peopleAssignments>?
     <~~htdp~~htc:isSkipable>...</~~htdp~~htc:isSkipable>?
     <~~htdp~~htc:expirationTime>...</~~htdp~~htc:expirationTime>?
     <~~htdp~~htc:attachments>...</~~htdp~~htc:attachments>?
   </~~htdp~~htc:humanTaskContext>
  </S:Header>
  <S:Body>
    ...
  </S:Body>
</S:Envelope>
```

Formatted: French (France)

Formatted: French (Canada)

Formatted: English (United States)

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

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

Formatted: German (Germany)

7.5 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 may 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 may signal whether or not the corresponding task may communicate with an invoking component via the WS-HT 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:

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

/~~htdp~~htp:HumanTaskAssertion

This policy assertion specifies that the sender of an input message 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.

/~~htdp~~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-HT coordination context MAY be passed with an input message. If the context is passed the receiving human task MUST be instantiated with the WS-HT protocol in place. The absence of the assertion is interpreted to mean that a WS-HT 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 `<~~htdp~~htp:HumanTaskAssertion>` policy assertion can also be used for notifications. In that case it means that the sender of an input message MAY pass the human task context information with the message. Other headers, including headers with the coordination context are ignored.

8 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 caller (i.e. a requesting application) to a human task may override static deployment information that may have been set.

8.1 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 human task 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-HT implementation 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.

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-HT implementation 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 human task MUST identify the instance of the caller. This can be done in two ways: First, the value of the [address] property may 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.

8.2 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>
    <~~htdp~~htcp:responseAction>xsd:anyURI</~~htdp~~htcp:responseAction>?
    <~~htdp~~htcp:responseOperation>xsd:NCName</~~htdp~~htcp:responseOperation>?
  </wsa:Metadata>
</wsa:EndpointReference>
```

Formatted: Portuguese (Brazil)

/wsa:EndpointReference/wsa:Metadata

This is a MANDATORY element of the EPR sent. 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/~~htdp~~htcp:responseAction

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

/wsa:EndpointReference/wsa:Metadata/~~htdp~~htcp:responseOperation

This element (of type `xsd:NCName`) specifies the name of the operation to be used by the receiving human task to send the response message from the human task back to the caller. The value of this element is taken from the `htd:~~call~~remoteTask/@responseOperation` attribute. If this element is specified the `<htcp:responseAction>` element MUST NOT be specified.

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 to be used 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, the callback information also specifies 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-HT implementation 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>
    <~~htdp~~htcp:responseAction>
      http://example.com/LoanApproval/approvalResponse
    </~~htdp~~htcp:responseAction>
  </wsa:Metadata>

```

Formatted: Portuguese (Brazil)

```
</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-HT implementation is defined using the `<wsdl:portType>` element. ~~i~~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 must be 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 ~~`<MyInstanceID`~~`<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>
    <~~yxz:~~xmp:MyInstanceID>42</~~yxz:~~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>

    <~~htdp~~htcp:responseOperation>approvalResponse</~~htdp~~htcp:responseOperation>

  </wsa:Metadata>

</wsa:EndpointReference>
```

8.3 Message Addressing Properties

Message addressing properties provide references for the endpoints involved in an interaction at the message level. For this case, WS-HumanTask 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 human task to send its response to.

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

The response message sent by the human task 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 is copied into the [action] property of the response message.
  - 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.
    - If no such `soapaction` attribute is provided, the value of the [action] property MUST be derived as specified in [WS-Addr-WSDL].
- Reference parameters are mapped as specified in [WS-Addr-SOAP].

8.4 SOAP Binding

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

The following is an example of a request message sent from the caller to the human task 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:~~htd~~htcp="~~http://www.example.org/WS-HT/protocol~~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>
```

```
        <~~yxz:~~xmp:MyInstanceID>42</~~yxz:~~xmp:MyInstanceID>
      </wsa:ReferenceParameters>
      <wsa:Metadata>
        <~~htdp~~htcp:responseAction>
          http://example.com/LoanApproval/approvalResponse
        </~~htdp~~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 human task 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 `<~~yxz:~~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>
    <~~yxz:~~xmp:MyInstanceID wsa:IsReferenceParameter='true'>
      42
    </~~yxz:~~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 human task 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 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 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 caller 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:~~htdp~~htcp="~~http://www.example.org/WS-HT/protocol~~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>
        <~~yxz:~~xmp:MyInstanceID>42</~~yxz:~~xmp:MyInstanceID>
      </wsa:ReferenceParameters>

      <wsa:Metadata>

        <wsdl:definitions
          targetNamespace="http://example.com/loanApproval"
          xmlns:wsdl~~11~~="..." 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>

        <~~htdp~~htcp:responseOperation>
          approvalResponse
        </~~htdp~~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 human task 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 `<`~~yxz:~~`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://www.example.org/WS-HT~~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>
    <~~yxz:~~xmp:MyInstanceID wsa:IsReferenceParameter='true'>
      42
    </~~yxz:~~xmp:MyInstanceID>
  </S:Header>
  <S:Body>...</S:Body>
</S:Envelope>
```

9 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 infrastructure.

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

10 Conformance

(tbd.)

11 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]

Web Services Coordination (WS-Coordination) Version 1.1, OASIS Committee Specification, February 2007, available via 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

A. Portability and Interoperability Considerations

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

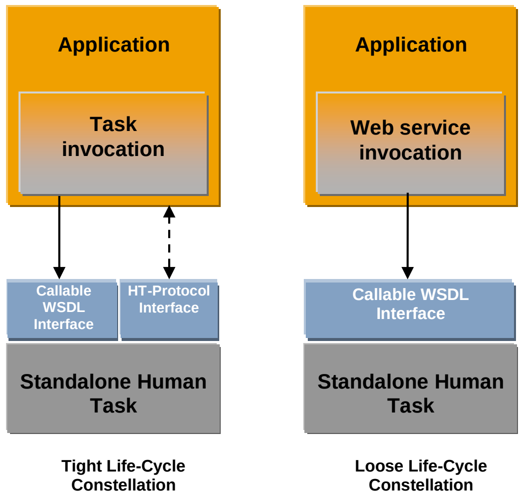
- Portability - The ability to take human tasks and notifications created in one vendor's environment and use them in another vendor's environment.
- Interoperability - The capability for multiple components (task infrastructure, task list clients and applications or processes with human interactions) 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 task infrastructure 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

Note to specification editors: the WS-HumanTask XML Schema definition is separately maintained in an artifact

```
ws-humantask.xsd
```

The contents of this artifact shall be copied back into this section before publishing the specification, e.g., as a committee draft.

Formatted: Bullets and Numbering

C. WS-HumanTask Data Types Schema

Note to specification editors: the WS-HumanTask data types XML Schema definition is separately maintained in artifact

```
        ws-humantask-types.xsd
```

The contents of this artifact shall be copied back into this section before publishing the specification, e.g., as a committee draft.

~~C.~~D. ~~Operations~~ WS-HumanTask API ~~WSDL~~Port Types

Formatted: Bullets and Numbering

Note to specification editors: the ~~WS-HumanTask API data types XML Schema definition, the WS-HumanTask API operation signature XML Schema definition, and the~~ WS-HumanTask API WSDL definition ~~are~~ is separately maintained in artifact~~s~~

~~ws-humantask-api.xsd~~

~~ws-humantask-api-wsdl.xsd~~

```
ws-humantask-api.wsdl
```

The contents of ~~these~~ this artifact shall be copied back into this section before publishing the specification, e.g., as a committee draft.

Formatted: Bullets and Numbering

E. WS-HumanTask Protocol Handler Port Types

Note to specification editors: the WS-HumanTask protocol WSDL definition is separately maintained in an artifact

```
        ws-humantask-protocol.wsdl
```

The contents of this artifact shall be copied back into this section before publishing the specification, e.g., as a committee draft.

Formatted: Bullets and Numbering

F. WS-HumanTask Context Schema

Note to specification editors: the WS-HumanTask context XML Schema definition is separately maintained in an artifact

```
        ws-humantask-context.xsd
```

The contents of this artifact shall be copied back into this section before publishing the specification, e.g., as a committee draft.

G. WS-HumanTask Policy Assertion Schema

Note to specification editors: the WS-HumanTask policy assertion XML Schema definition is separately maintained in an artifact

```
      ws-humantask-policy.xsd
```

The contents of this artifact shall be copied back into this section before publishing the specification, e.g., as a committee draft.

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

~~D.~~H. Sample

This appendix contains the full sample used in this specification.

WSDL Definition

Note to specification editors: the WS-HumanTask example WSDL definition is separately maintained in an artifact

```
ws-humantask-example-claim-approval.wsdl
```

The contents of this artifact shall be copied back into this section before publishing the specification, e.g., as a committee draft.

Human Interaction Definition

Note to specification editors: the WS-HumanTask example Human Task definition is separately maintained in an artifact

```
ws-humantask-example-claim-approval.tel
```

The contents of this artifact shall be copied back into this section before publishing the specification, e.g., as a committee draft.

~~E.Schema of Protocol Messages~~

Formatted: Bullets and Numbering

~~Note to specification editors: the WS-HumanTask protocol XML Schema definition is separately maintained in an artifact~~

~~ws-humantask-protocol.xsd~~

~~The contents of this artifact shall be copied back into this section before publishing the specification, e.g., as a committee draft.~~

~~F.Protocol Handler Port Types~~

~~Note to specification editors: the WS-HumanTask protocol WSDL definition is separately maintained in an artifact~~

~~ws-humantask-protocol.wsdl~~

~~The contents of this artifact shall be copied back into this section before publishing the specification, e.g., as a committee draft.~~

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

~~G. Schema of the Task Context~~

~~Note to specification editors: the WS-HumanTask context XML Schema definition is separately maintained in an artifact~~

```
~~ws-humantask-context.wsdl~~
```

~~The contents of this artifact shall be copied back into this section before publishing the specification, e.g., as a committee draft.~~

Formatted: Bullets and Numbering

~~H.~~I. Acknowledgements

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

Members of the BPEL4People Technical Committee:

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
James Bryce Clark, OASIS
Luc Clément, Active Endpoints, Inc.
Manoj Das, Oracle Corporation
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
Ashok Malhotra, Oracle Corporation
Mike Marin, IBM
Mary McRae, OASIS
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.
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:

Formatted: English (United States)

Formatted: English (United States)

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

Formatted: English (United States)

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.

~~I.~~J. Non-Normative Text

Formatted: Bullets and Numbering

~~J.~~K. 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